NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

PARALLEL PROCESSING PERFORMANCE EVALUATION OF MIXED T10/T100 ETHERNET TOPOLOGIES ON LINUX PENTIUM SYSTEMS

by

Steven W. Decato

March 1997

Thesis Advisor: Bert Lundy

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE TITLE OF THESIS: Parallel Processing Performance Evaluation of Mixed T10/T100 Ethernet Topologies on Linux Pentium Systems			5. FUNDING NUMBERS
6. AUTHOR(S) Steven W. Decato			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

The intent of this thesis is to answer the question as to whether real-time battlefield visualization, once requiring high-speed UNIX workstations and specialized parallel processors, can now be now performed on relatively inexpensive off-the-shelf components.

Alternative network topologies were implemented using 10 and 100 megabit-per-second Ethernet cards under the Linux operating system on Pentium based personal computer platforms. Network throughput, processor and video performance benchmark routines were developed to assess the hardware's potential for parallel application in a distributed environment. Code was first ported to the Linux environment. Benchmark routines were then developed and tested on various machines.

Dual 200 MHz Pentium Pro processor performance exceeded the dual processor 50 MHz SUN and 40 MHz SGI UNIX workstations currently used for terrain generation by a factor of 30 using a simple ray trace algorithms as a basis for comparison. The Intel Pentium Pro personal computer proved to be a capable platform for generating six to ten frame-per-second terrain simulations. However, Fast Ethernet throughput averages only 2.5 megabytes-per-second, thereby limiting the usefulness of a distributed approach designed to increase performance by dividing workload across the network.

14. SUBJECT TERMS Simulation, Perspective View Generation, Benchmarks, Performance, Parallel, Clusters, MPI			15. NUMBER OF PAGES 84
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

PARALLEL PROCESSING PERFORMANCE EVALUATION OF MIXED T10/T100 ETHERNET TOPOLOGIES ON LINUX PENTIUM SYSTEMS

Steven W. Decato
Major, United States Army
B.A., University of South Florida, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1997

ABSTRACT

The intent of this thesis is to answer the question as to whether real-time battlefield visualization, once requiring high-speed UNIX workstations and specialized parallel processors, can now be now performed on relatively inexpensive off-the-shelf components.

Alternative network topologies were implemented using 10 and 100 megabit-per-second Ethernet cards under the Linux operating system on Pentium based personal computer platforms. Network throughput, processor and video performance benchmark routines were developed to assess the hardware's potential for parallel application in a distributed environment. Code was first ported to the Linux environment. Benchmark routines were then developed and tested on various machines.

Dual 200 MHz Pentium Pro processor performance exceeded the dual processor 50 MHz SUN and 40 MHz SGI UNIX workstations currently used for terrain generation by a factor of 30 using a simple ray trace algorithms as a basis for comparison. The Intel Pentium Pro personal computer proved to be a capable platform for generating six to ten frame-per-second terrain simulations. However, Fast Ethernet throughput averages only 2.5 megabytes-per-second, thereby limiting the usefulness of a distributed approach designed to increase performance by dividing workload across the network.

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author gratefully acknowledges the support of the following agencies for equipment, data and funds for the development of related software packages without which this work would not have been possible.

Texcom Experimentation Center
Fort Hunter Liggett
Jolon, CA 93928

TRADOC Analysis Command-Monterey
Naval Postgraduate School
Monterey, CA 93943

I. INTRODUCTION

A. PURPOSE

The purpose of this research paper is to present the evaluation of Personal Computers (PC) clusters as a potential low cost alternative to UNIX Workstations for supporting real-time terrain modeling simulations. Our intent was to explore various network communications protocols and determine maximum network throughput as a basis for a parallel implementation of the PEGASUS Perspective View Generator. Benchmark programs were developed and results analyzed as a function of network topologies, number of processors, and link communication speeds. Although a majority our work involved analysis of network performance, benchmark programs were also written to evaluate processor and video performance. Research included exploring the difficulties in selecting and installing appropriate hardware using the Linux operating system. Porting problems encountered in moving software from the UNIX workstation environment to an Intel PC based Motif/X Window platform were examined. The intent of this analysis is to answer the question of whether real-time battlefield visualization, once conducted in the lab on high-speed workstations and specialized parallel processors, can now be now performed on relatively inexpensive off-the-shelf components.

B. BACKGROUND

In 1983 when the Ada programming language was first developed, the IBM Personal Computer XT (PC) had been in production just two years and was already being replaced by the IBM PC/AT. In the same year Rick Mascitti coined the name C++ for Bjarne Stroustrup's "C with Classes". The Intel 80286 was the most advanced Intel CPU on the market having been introduced in 1982. In just fifteen years, performance of these low cost processors has blurred the computational benefits that have typically separated UNIX workstations from the personal computer.

Today the development of high-speed local area networks, low cost CPUs running with clock speeds in excess of 200 megahertz, and the proliferation of inexpensive Personal Computers (PC) have expanded our vision of a distributed system. The possibility of thousands of networked computers working concurrently on a single task or cooperatively on many tasks is within our reach. In today's climate of *right sizing*, smaller budgets are forcing purchasers to leverage more performance out of existing hardware. A PC based distributed approach offers the promise of low cost scalable performance improvements.

The personal computer has also become a popular game platform due to recent gains in video display performance. Scores of simulation-type games put the player in the cockpit or on the ground navigating through a virtual world in real-time. Although video realism has improved dramatically through texturing, these games have done little in the way of placing the user in a photo-realistic and geographically accurate environment.

This area of research is not limited by available geographical data. The United States Geological Survey is conducting a National Mapping Program (NMP) that includes the production of digital cartographic data and graphical maps. Figure 1 illustrates the regions of the United States already included in the digital survey.

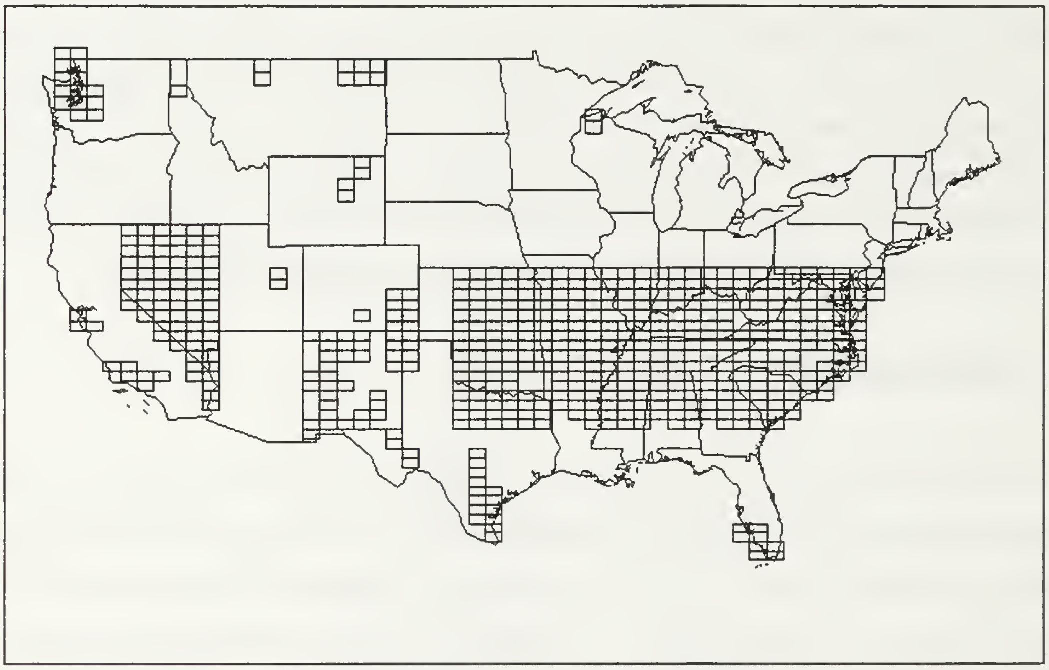


Figure 1. National Mapping Program Progress To Date

Researchers have been working in this area for several years at the US Army Test Facility at Ft. Hunter Liggett, California. They have developed a high fidelity, terrain accurate, simulation system known as PEGASUS. Starting with low-resolution satellite imagery and the corresponding terrain data from the Defense Mapping Agency, this system adds additional high-resolution photographic data sources to generate a terrain database accurate to one-meter resolution.

With the closing of Ft. Hunter Liggett, work proceeded at the Naval Postgraduate School to build world-wide database structure along with tools for integrating standard Defense Mapping Agency products (Baer, 1995). A perspective view generator was also developed which included an X-Window interface operating on a Silicon Graphics workstation.

A parallel research effort conducted between the Naval Postgraduate School, TRADOC Analysis Command Monterey, and US Army TEXCOM was started to integrate all the legacy code developed on the SUN, Silicon Graphics, and Transputer platforms into a single low cost rapid terrain generation and perspective view generation workstation. This project, code named TELLUS, ported existing code from a proprietary parallel processor machine to a relatively portable X-Window environment. The work focused on extending the database generation procedures to lower resolution formats, thereby providing the capability of feeding virtual and constructive simulators.

One underlying requirement was to identify the feasibility of moving the application to more powerful computer hardware capable of supporting the processing needs of realistic battlefield replication and rapid database generation without a significant cost increase. The most likely low cost candidates are systems built from commercial PC components. Personal computers have historically been unable to provide the performance necessary to support computationally intensive applications such as PEGASUS. Thus expensive workstations and specialized equipment were necessary to achieve reasonable performance.

We believe the recent performance improvements will allow low cost commercial components to provide the computing power required for high-resolution terrain database construction and real-time perspective view generation. To test this hypothesis we have constructed a test-bed using two Pentium based PCs, a fast 100 BASE-T Ethernet operating under the Linux operating system and several 486 PCs. Using this equipment, we developed benchmark and test routines to estimate the performance of our simulation application. After executing the benchmarks outlined in this thesis we find the results are extremely encouraging.

This report provides further details on the configuration, software environment, and benchmark tests performed to explore the suitability of using off-the-shelf PC equipment for parallel high-resolution battlefield simulation.

C. RESEARCH QUESTIONS

1. Primary Research Question

To what extent can the PCI bus and new high performance peripherals deliver the performance necessary for battlefield simulation applications?

2. Secondary Research Questions

- What other performance considerations affect the suitability of the Personal Computer as a prospective platform for high- speed terrain modeling?
- Can a distributed approach enhance performance of the Perspective View Generator?

D. SCOPE

This thesis focuses primarily on the use of Intel based Personal Computer using the Linux operating system. In the interest of time and expense we chose not to examine other compatible processors such as those manufactured by DEC, AMD and Cyrix. Linux was chosen as the target operating system because of its low cost and relatively minor differences between the UNIX variants running on high-end workstations.

E. METHODOLOGY

To achieve our goal, the research was divided into several tasks. First, hardware was purchased to host the Linux operating system. Linux was then installed and the network put into operation. Research was conducted to evaluate the availability and applicability of various communications Application Programming Interfaces (APIs). Once the APIs were chosen, various benchmark programs were written in C to test system performance. Finally, the results were recorded and analyzed to evaluate our research questions.

F. ORGANIZATION

Chapter II (Building a Network) provides an overview of the hardware acquired for the project. A discussion of the PC hardware architecture that impacts on performance of the Perspective View Generator performance is presented. Chapter II also examines processor, video and network limitations.

Chapter III (Performance Analysis) presents the benchmark methods and programs developed to test PC performance using TCP/IP sockets and streams, FTP file transfers, and the Message Passing Interface (MPI) programming environment.

Chapter IV (Summary, Conclusions and Recommendations) summarizes the findings of the research, answers the research questions, and presents recommendations for further research and study.

G. BENEFITS OF STUDY

Personal computer cost is a fraction of UNIX workstation cost. The benefits to the training and intelligence community are equally exciting. Given relatively recent data from the battlefield, such a system could provide intelligence analysts, pilots or soldiers the ability to safely walk or fly over terrain in a virtual environment. Near real-time intelligence data could also populate the view with enemy target locations and provide three dimensional battlefield visibility. Vehicles, equipment, and even aircraft could be dynamically tracked over an accurate view of the terrain based on digitized photographic and elevation data available from a multitude of sources. Graphical overlays could be placed over the terrain to give planners a feel for boundaries and identify key troop and re-supply locations. Additionally, the effects of current atmospheric conditions could be added to the scene to produce even more realistic images based on time and weather conditions.

This thesis provides the building blocks for analyzing networked PC system performance. Our research constructs a performance model of a personal computer hosted system that we hope will soon be capable of handling such a scenario.

II. BUILDING A NETWORK

A. HARDWARE CONFIGURATION

The TELLUS configuration is expected to support three main functions:

- **Analysis Functions** - requiring standard low cost report generation, communication, mathematical and presentation tools.

- **Database Generation-** requiring multiple seats, large image storage capacity and emphasizing graphic interactivity.

- **Battlefield Simulation-** requiring super computer performance processing, real-time communication, and single function operation

The networked PC based design will eventually support all of these. The personal computer has historically supported the first function, serving as a relatively low cost, low speed platform for business applications with abundant software tools that have made the PC so successful. We have only recently turned to the personal computer for scientific database generation and battlefield simulation due to the emergence of fast 32-bit processors connected to high-speed communications devices. Multiple processors can now share large image databases during interactive database creation. Likewise, these same processors may soon join forces and provide the processing power to support realistic real-time battlefield simulation.

Figure 2 shows a block diagram of the design configuration for the TELLUS project. Although only two processors are shown in detail, the intent was to design a scalable networked system. The diagram is functionally organized with input devices on the left, computational network power in the center, and output devices on the right. This diagram represents the minimum system that can be used to explore networked parallel processing applications and act as controller/peripheral device pair interfaces to the bulk of the computing power contained in the expanded network.

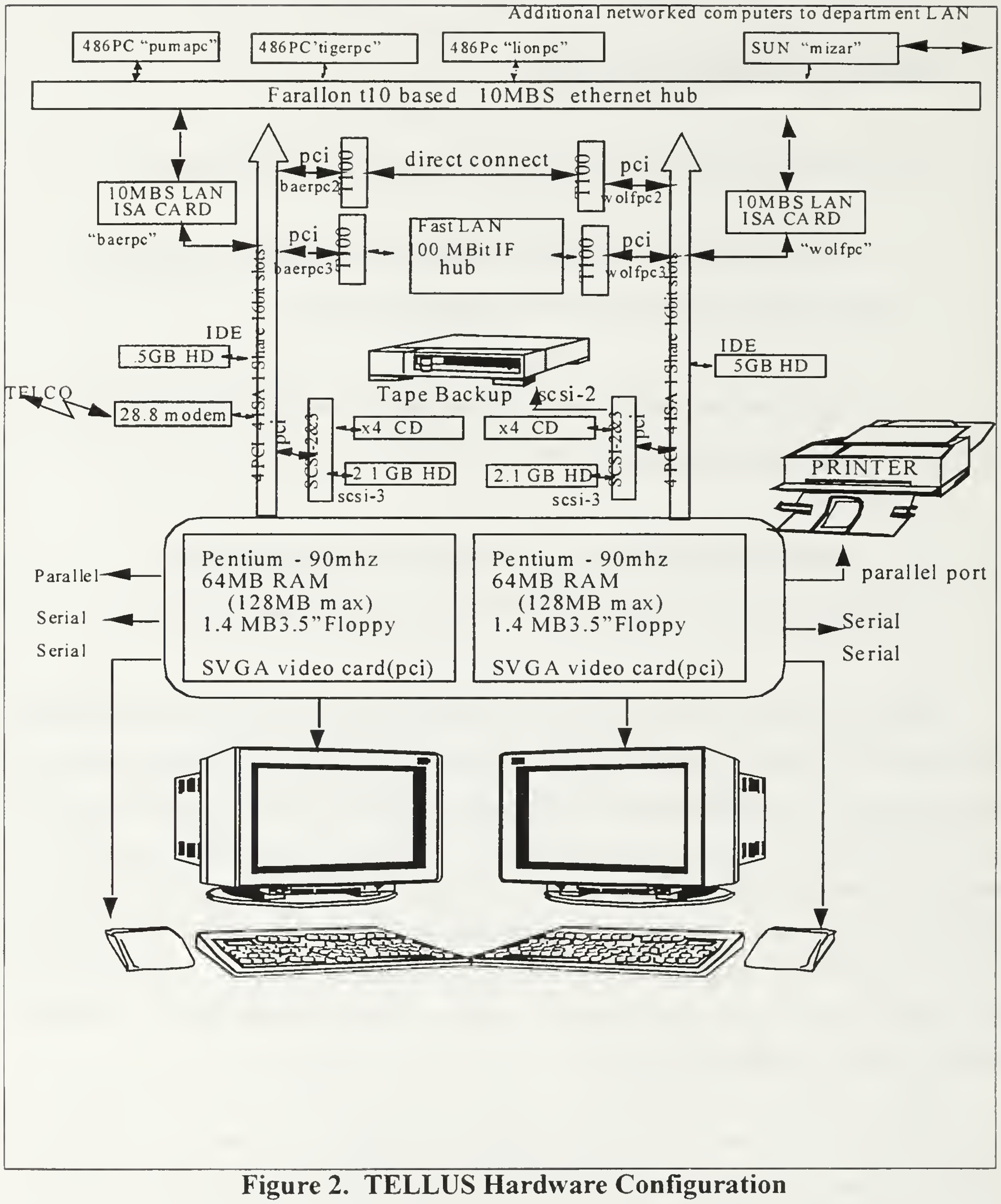


Figure 2. TELLUS Hardware Configuration

Cost is the main attraction of the PC based configuration. A high-end personal computer system, such as the Pentium 90 network machine purchased for the project, has remained a constant $3000 since the IBM PC was first introduced. Only the capabilities and performance have increased. In November of 1995 when we first considered purchasing equipment for this research, the Pentium 133 was the fastest personal computer on the market. Since then processor performance has doubled, memory prices have plummeted from $35 per megabyte to $5 megabyte and IDE hard drive prices have dropped from $200 per megabyte to $100 megabyte. We believe interface nodes with peripherals will continue to cost about $3000 while network computational nodes will cost under $1000 each.

B. THE OPERATING SYSTEM

We chose the Linux/PC combination for the first attempt at a PC port of PEGASUS. This UNIX-like operating system has a large following among researchers and computer science students alike. Its ability to compile and run source code intended for UNIX Workstations is especially attractive. Furthermore, Linux is available at little or no cost. In contrast, SCO UNIX is commercially available for $400 per node. This additional cost seems prohibitive since it substantially increases total system cost as nodes are added. Linux provides the same functionality with no requirement for additional licensing fees as nodes are added. The decision to use Linux had one major drawback. When our research first began, not a single book existed on Linux in any local bookstore. Fortunately this information void has improved. Most major bookstores now have several books devoted to the operating system. Even Motif has been ported to the PC to replace the barren X Window desktop.

Our first task was to obtain Linux and install it on a network of five computers; two Pentium 100 systems and three 486-66 systems. During the installation phase, we had a variety of hardware problems to contend with. Drivers were difficult to locate for our mixture of SCSI and IDE storage subsystems. Without a CDROM drive, we had to download Linux from Internet sites one floppy diskette at a time. The Pentium systems utilized Adaptec 2940W SCSI cards not directly supported under Linux at the time. The first of many challenges was to obtain a beta version of the driver mentioned briefly in an obscure HOWTO document on the Linux distribution CDROM. Once the source code for the driver was obtained, it was compiled into the operating system kernel.

Maintaining a Linux network is time intensive and a thorough knowledge of UNIX system administration is required. Those of us with little experience in this area found that Linux presented an administrative burden that sometimes interfered with research. This is true of any UNIX implementation.

C. THE PROGRAMMING INTERFACE

Once the network was running under Linux, we scoured the Internet searching for available message passing libraries to ease the development of benchmark programs that would be used to test the potential speed of the network. The two most popular Application Programming Interfaces (APIs), TCP/IP sockets and streams, were included with Linux. We were able to find many sources of information on the topic. Benchmarks programs were quickly developed and tested.

After evaluating the available libraries, we settled on the LAM port of Message Passing Interface for our secondary tests. An MPI standard has been established and subsequently ported to many platforms. For this reason, MPI appeared attractive as an interface that could be ported from one environment to another with relatively minor modifications.

D. PARALLELISM AND CLUSTERING

Perspective View Generation is a computationally intensive task requiring millions of calculations per second to support smooth frame rates. We were attracted to the idea of dividing the workload across network clusters in hopes that dividing the problem would result in a faster screen refresh rates. The serial version of the program had already been optimized to the maximum extent possible.

The notion of a network of computers working together is defined as a *distributed system*. A distributed system is an interconnection of one or more processing nodes (a system resource that has both computational and storage capabilities), and zero or more storage nodes (a system resource that has only storage capabilities, with the storage addressable by one or more processing nodes)". (ISO/IEC 8652, 1995) Distributed systems differ from centralized systems in that centralized systems consist of a single CPU, its memory, peripherals, and some terminals.

"A *distributed program* comprises one or more partitions that execute independently (except when they communicate) in a distributed system." (ISO/IEC 8652) If we accept Booch's definition of an embedded system as it applies to multiple networked CPUs working together concurrently (Booch, 1987), can this technique provide benefits to a PC based perspective view generator?

Andrew Tanenbaum divides Flynn's Multiple Instruction Stream, Multiple Data Stream model (Flynn, 1992) into two subcategories: Multiprocessors that share memory and multi-computers having private memory (Tannenbaum, 1992). We will limit our discussion to a bus-based multi-computer comprised of two or more workstations connected to a Local Area Network (LAN) using Fast Ethernet.

1. The Central Processing Unit (CPU)

The decision to build applications on networked clusters requires an understanding of the performance improvement expected from single processors. The time and effort it takes to restructure an application using a parallel design is often futile as single node performance improves faster than the development effort expended to achieve modest gains. This section provides insight into current processor technology that guided our purchasing decisions.

The Pentium Pro is Intel's latest and most powerful processor. While currently shipping versions are clocked at 200 megahertz, Intel has demonstrated a 300-megahertz Pentium. Tests show that the 300 megahertz chip delivers 50% better performance than Intel's top Pentium Pro. The new chip won't be available for home PCs until late summer 1997. Sources at the International Solid State Circuits Conference also report that Intel's processors have reached clock speeds of 433 megahertz and 451 megahertz in the laboratory. The 233-megahertz and 266 megahertz Pentium MMXs will reportedly be out by early summer 1997 as well.

Digital Equipment's 500mhz 21164 is currently the fastest CPU installed in personal computers today. The recent growth of Microsoft Windows NT as the software of choice for server platforms has hindered the Alpha's acceptance. Although Microsoft has continued to provide a native version of NT for the Alpha platform, software developers have focused development effort on Intel specific applications.

Digital has responded recently by releasing **FX!32**, an Intel Windows emulator capable of directly executing Intel binaries. Unfortunately, as with most emulators, it does not provide the performance the 500mhz 21164 is capable of. Furthermore, it is not 100% compatible with all Intel based Windows binaries.

The 21164 is twice as fast as Intel's Pentium Pro 200 when comparing integer operations, and nearly four times faster when comparing floating-point operations.

Figure 3 compares integer and floating point performance of the Alpha and Pentium Pro processors using the SPEC95 benchmark suite discussed in the next section.

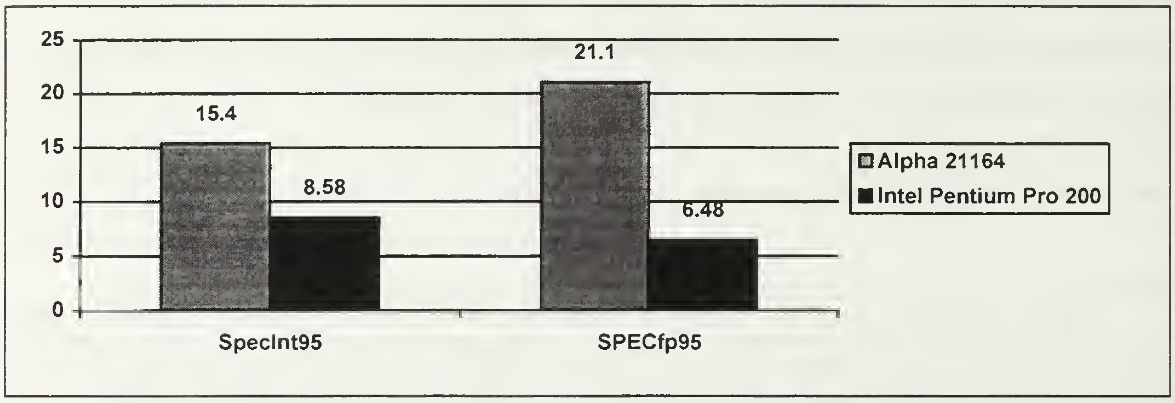
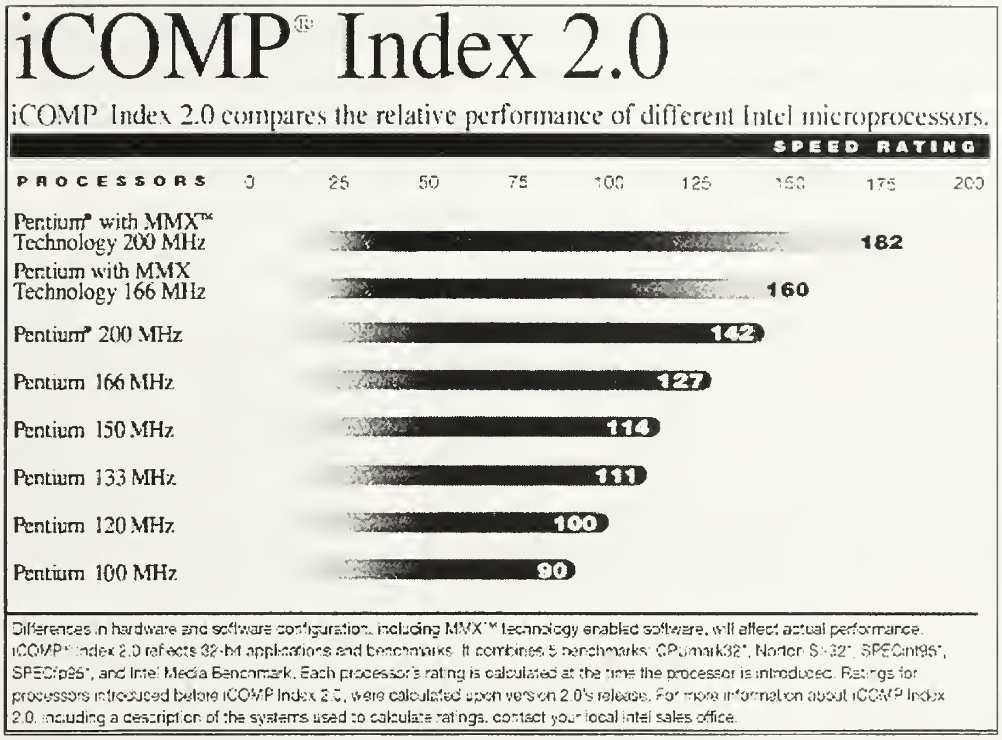


Figure 3. SPEC Benchmark Comparison of Alpha and Intel CPUs

Measuring processor performance can be controversial. There are many benchmark software products useful for measuring specific capabilities of a system, but reliance on any one suite can mislead potential system buyers. Identifying how benchmark results relate to a user's computing needs is a difficult but essential part of the system acquisition process. Often manufacturers will design their own benchmark programs. The results are then published as marketing tools for touting their products. These numbers may have little relevance to overall system performance in actual applications.

2. Intel Comparative Microprocessor Performance Benchmark (iCOMP)

Intel publishes its own iCOMP (Intel Comparative Microprocessor Performance) index. They are forthright in admitting that their only intent is "to help end users decide which Intel microprocessor best meets their desktop computing needs." The iCOMP Index 2.0 rating is based on the integer, floating point and multimedia capabilities. Figure 4 demonstrates the near linear increase in performance in Intel's Pentium processor line using the iCOMP index.



Differences in hardware and software configuration, including MMX™ technology enabled software, will affect actual performance. iCOMP® Index 2.0 reflects 32-bit applications and benchmarks. It combines 5 benchmarks: CPUmark32*, Norton SI-32*, SPECint95*, SPECfp95*, and Intel Media Benchmark. Each processor's rating is calculated at the time the processor is introduced. Ratings for processors introduced before iCOMP Index 2.0, were calculated upon version 2.0's release. For more information about iCOMP Index 2.0, including a description of the systems used to calculate ratings, contact your local Intel sales office.

Figure 4. Intel's iCOMP Index

3. The Standard Performance Evaluation Corporation (SPEC)

A more neutral benchmark suite may be appropriate when comparing general performance between one system and another. SPEC is a non-profit corporation formed to "establish, maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers" (from SPEC's bylaws). The founders of this organization believe that the user community will benefit greatly from an objective series of applications-oriented tests designed to serve as common reference points and be considered during the evaluation process. SPEC publishes vendor benchmark results on its World Wide Web home page at http://www.specbench.org *(SPEC, 1997)*.

SPEC95 is the current suite of benchmarks published by SPEC. These benchmarks measure the performance of the processor, memory system, and compiler code generation. UNIX is normally used as the portability vehicle, but the benchmark programs have been ported to other operating systems as well. The percentage of time spent in operating system and I/O functions is generally negligible. SPEC95 CPU benchmarks are internally composed of two collections: CINT95 and CFP95.

a. CINT95

CINT 95 is a suite of integer based programs representing the CPU-intensive part of system or commercial application programs. Figure 5 illustrates the relative integer performance of the Intel Pentium series processors. (SPEC, 1997) Notice that the performance gains follow the clock rating of the chip itself. Like the Intel's iCOMP rating, increases in performance since the introduction of the Pentium 100 have been nearly linear. However, the recent introduction of the Pentium MMX series has pushed performance of the 166 megahertz Pentium MMX beyond that of the Pentium 200.

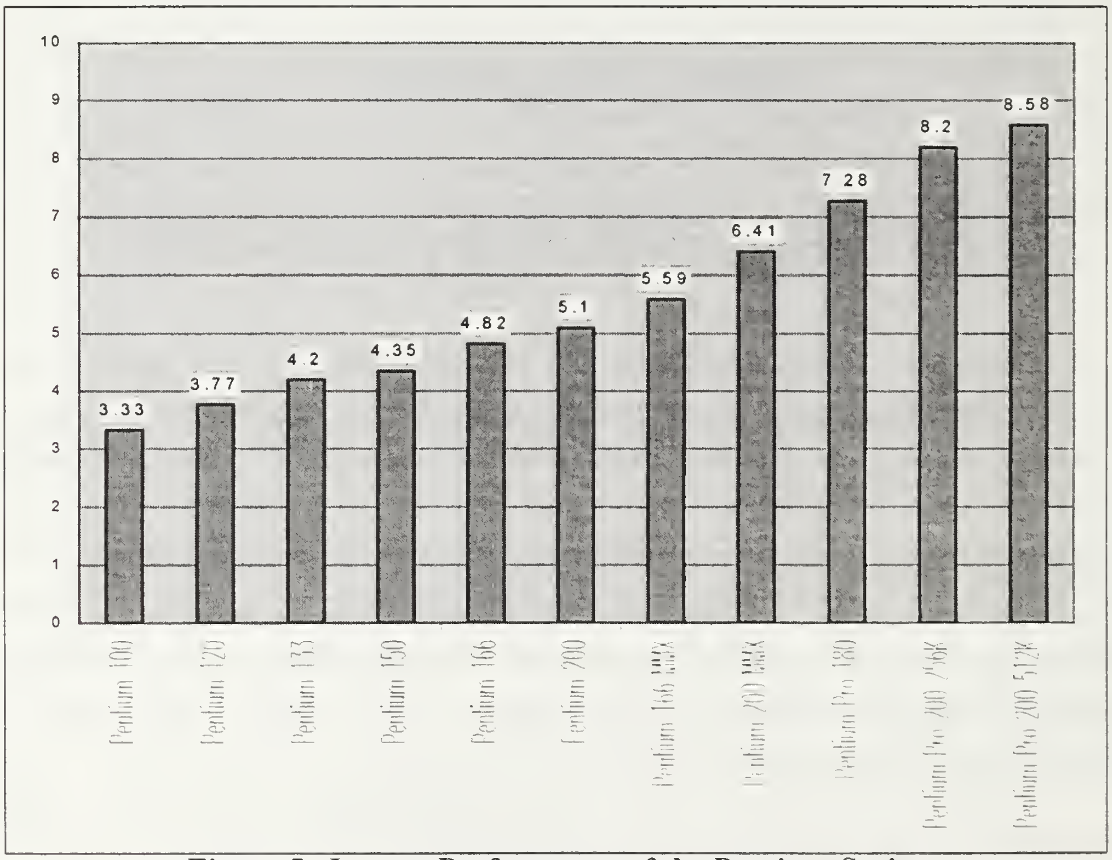


Figure 5. Integer Performance of the Pentium Series

Intel's top performers, the 200 megahertz Pentium Pro is still champion of the average desktop system, but figure 6 shows that there is no substitute for higher clocked processors to boost performance (SPEC, 1997). The increase in performance is not linear based on clock speed alone. Other factors play a role in overall performance including on board cache size, RISC versus CISC architecture, Bus design and high-speed peripherals.

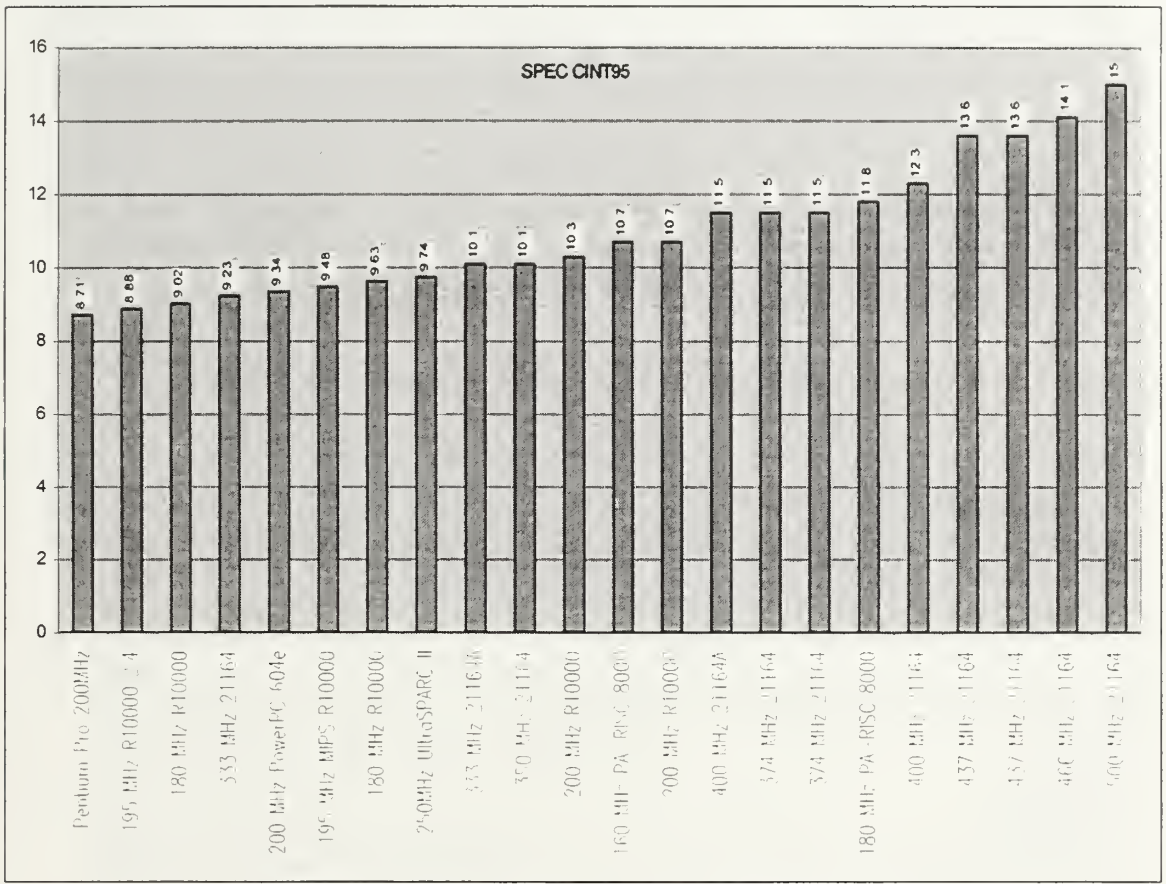


Figure 6. Integer Performance of Various CPUs

b. CFP95

Likewise, CFP95 measures the CPU-intensive part of numeric-scientific application programs. When compared with other processors, Intel lags well behind the competition in floating point performance. As stated previously, the Pentium Pro is eclipsed by Digital's 500 megahertz 21164. When examining the benchmarks presented in figure 7 (SPEC, 1997), it is obvious that performance is sacrificed at the expense of cost. Further research should be conducted to determine if the faster Alpha processors see relative increases throughout the architecture. We have seen bottlenecks that appear to be Bus related. Moving to a faster CPUs may not reap the performance benefits indicated by the benchmarks.

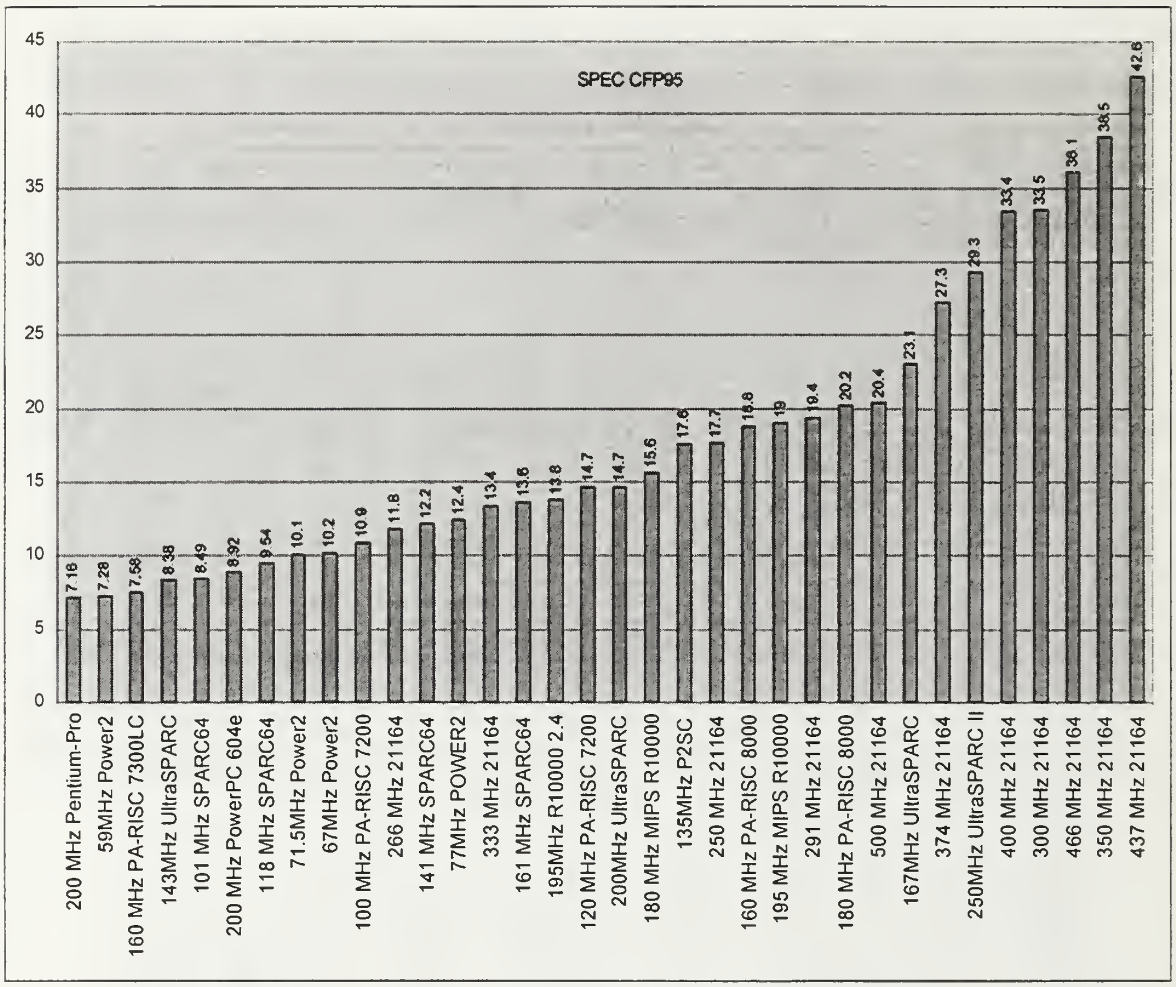


Figure 7. Floating Point Performance Comparison of Various Processors

c. SPEC Rate

With this measurement method, called the "homogeneous capacity method", several instances of a given benchmark are executed. This method is particularly suitable for multiprocessor systems. The results, called SPEC rate, express how many jobs of a particular type (characterized by the individual benchmark) can be executed in a given time. The SPEC rates therefore characterize the capacity of a system for compute-intensive jobs of similar characteristics. Two benchmark programs make up the "Rate" suite. SPECInt Rate measures the integer capacity of a system while SPECfp Rate measures the floating-point capacity (SPEC, 1997).

d. SPECint_rate95

Sometimes adding an extra processor can make up for lack of computational power. Figure 8 demonstrates that characteristic (SPEC, 1997). In fact, a dual Pentium Pro 166 machine is about as fast as the envied 500 megahertz Alpha 21164.

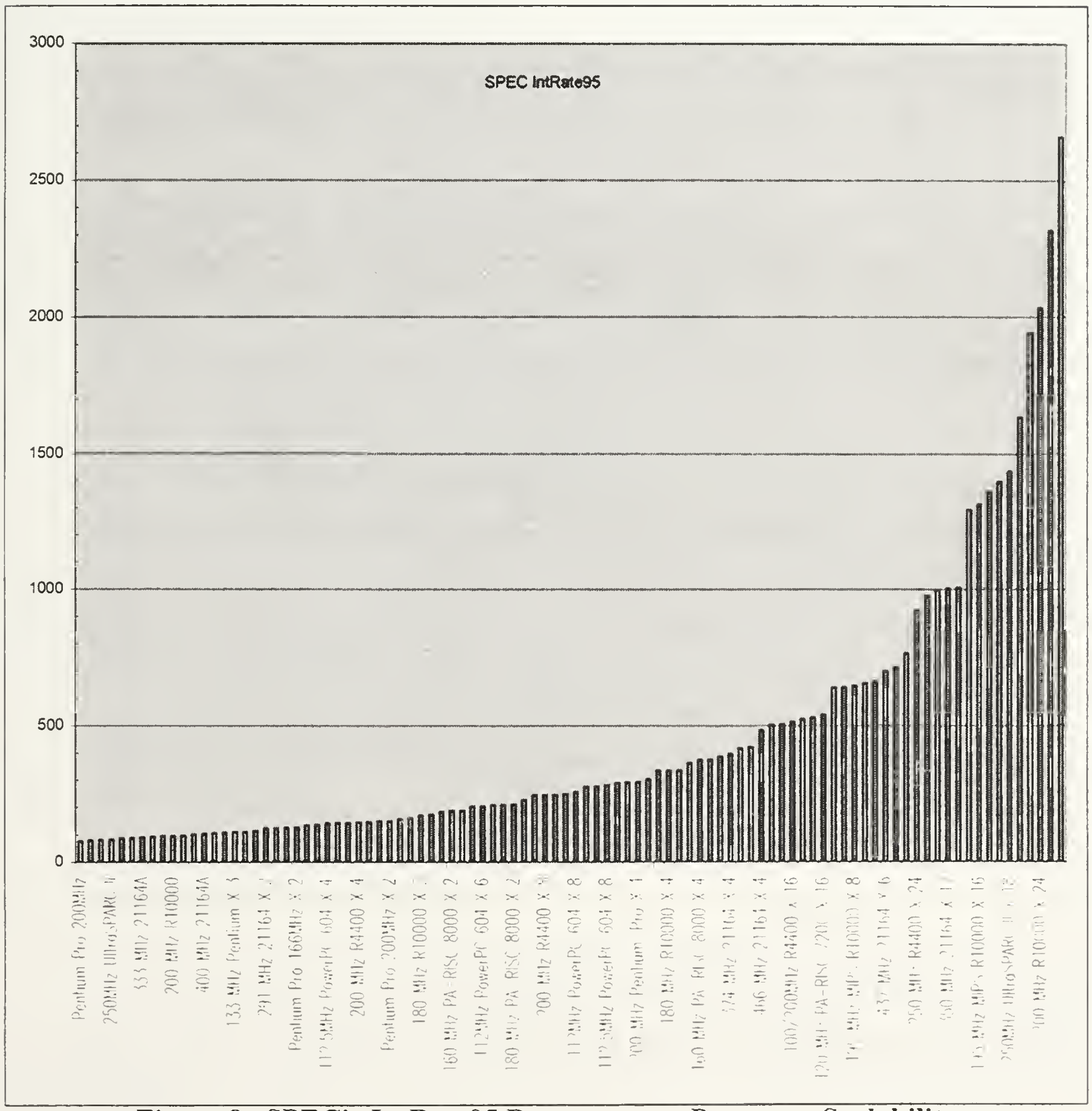


Figure 8. SPEC's IntRate95 Demonstrates Processor Scalability

III. PERFORMANCE ANALYSIS

Our approach has been to define critical functions, write small benchmark programs designed to test these functions, and finally compare the execution times between workstations upon which the PEGASUS software is currently implemented with those achieved in the PC configuration.

The three main areas selected for testing are:

- **Communications**: How fast can messages and data arrays be sent from one processor to another?

- **Image display**: How fast can an image in main memory is transferred to the display?

- **CPU and Main memory**: How fast can the inner ray-trace loop and terrain arrays be accessed?

In addition to these questions, there are disk performance issues requiring the quantification of the wide 16 bit SCSI drives, the impact of swap space, the effect of main memory size, and a host of issues that can also effect performance. We have concentrated on the three areas mentioned above because in the past personal computers have not performed well enough in these areas to be used instead of workstations. Until these performance hurdles had been overcome, further detailed investigation and optimization was pointless.

A. COMMUNICATION TESTS

At the time of writing, four 3COM 100 bit per second communications cards have been installed under Linux and several communication tests performed and are outlined in this section.

Figure 9 shows comparative benchmarks published in trade journal literature for 100BT-card operation. According to these numbers, close to a megabyte per second per link can be expected from a 5-client cluster using Novell's NetWare 4.02 on a HP NetServer 4/100 LC (LAN Times Magazine, 1995). A single client transfer rate ranges from 2.5 to 7Mbps. This is a long way from 100Mbits/s raw hardware speed.

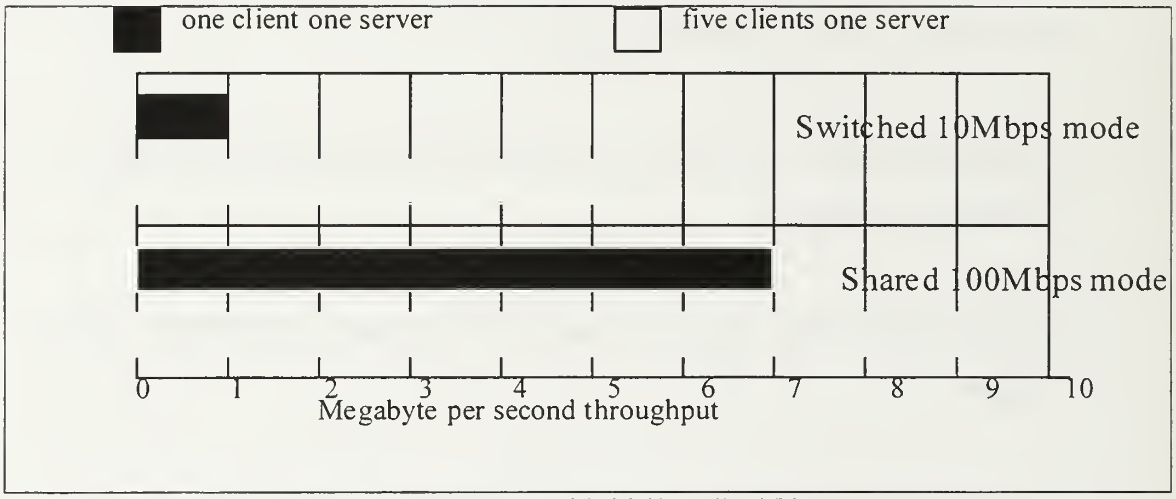


Figure 9. EtherExpress 10/100 Fast LAN Performance

Our research indicates these numbers are overoptimistic. Considerably slower speeds can be expected in a reliable contention-less domain. The use of a Fast Ethernet in our test systems utilizing a small number of processors achieved performance on par with the 1.2 megabyte-per-second communication speeds achieved using the current Transputer Link real-time perspective view generators. This speed is not considered adequate for smooth parallel frame rate operation. However, it is adequate to support several frames per second and provide playback speeds comparable to those achieved more expensive equipment.

The following tests were conducted to explore performance gained using other communications methods:

1. FTP File Transfer Test

Before designing any low level benchmark programs to test potential network performance, we decided to use available utilities to establish rough performance figures

2. Local File Copy Test

To test the speed that data could be moved local from one region of a disk to another, the following script was executed:

```
time cp gcc-2.7.2.bin.tar .. (10,752K)
time cp lam60.tar ..(3006K)
time cp lam60.tar.gz ..(522K)
time cp linux-2.0.8.tar .(24,187K).
time cp linux-2.0.8.tar ..
time cp lam60.tar.gz ..
time cp lam60.tar ..
time cp gcc-2.7.2.bin.tar ..
```

Figure 12 indicates the performance achieved from executing the previous script. These numbers were also repeatable and did not change even if copies were made from the SCSI to the IDE drive or from SCSI to SCSI. Approximate results are quite unambiguous. Hence the communication is between 1000K and 4100K bytes/sec with the average being 2400K per second. The local transfer rate is up to ten times that of the 10BASET capacity and as much as twice as fast and 100BT network transfers.

```
0.01user 1.55system 0:02.57elapsed 60%CPU (4100K throughput)
0.03user 0.45system 0:01.11elapsed 43%CPU (2700K throughput)
0.01user 0.08system 0:00.12elapsed 75%CPU (4350K throughput)
0.07user 3.71system 0:12.15elapsed 31%CPU (1000K throughput)
0.04user 3.61system 0:06.59elapsed 55%CPU (3640K throughput)
0.01user 0.09system 0:00.26elapsed 38%CPU
0.02user 0.49system 0:00.62elapsed 82%CPU
0.02user 1.90system 0:08.35elapsed 22%CPU (2400K average)
```

Figure 12. Linux Local Disk Copy Transfer Rate Samples

3. X-Server To Remote Machine

An X-server program **viewtest** was written to test the speed with which an X-server could display images in this configuration. The program sets up two image buffers and sequentially calls the server with the subroutine outlined below:

```
XPutImage(XtDisplay(window->w), XtWindow(window->w), window-
>gc,window->image, 0, 0, 0, 0, w, h);
```

This program was run using a server on a local host **baerpc** and a server on a remote host wolfpc. A session dialogue follows:

NOTE: 65536 bytes/frame * 100 frames/sec = 6.55 Mbytes/sec

```
baerpc:~/BENCH_MARK/PIXELS$ viewtest -display wolfpc:0.0
Generating pattern 1...
Generating pattern 2...
PRESS RETURN TO START BENCH MARK...
Total time took 3 seconds to display 100 frames
Average frame rate was 33 per second
NOTE: 65536 bytes/frame * 33 frames/sec = 2.16 Mbytes/sec

baerpc:~/BENCH_MARK/PIXELS$ viewtest
Generating pattern 1...
Generating pattern 2...
PRESS RETURN TO START BENCH MARK...

Total time took 1 seconds to display 100 frames
Average frame rate was 100 per second
```

ANALYSIS: X-server takes 1/6.55 =	1526 sec/Mbytes to display on local host
X-server takes 1/2.16 =	4629 sec/Mbytes to display remotely
The difference.	3103 sec/Mbytes attributed to Comm delay

RESULT: steady state transfer rate is 3.222 Mbytes/sec

This result is quite fast, repeatable to a factor of 2 and of unknown reliability. The window observed would flip by quickly. Consequently, there was no chance to verify the detailed accuracy of the image displayed. The issue of reliability is discussed in the following section. Raw transfer rates of 3Mbytes/sec represent approximately 50% of the bare hardware speed and may easily be close to maximum throughput achievable for steady state operation.

4. Socket Transfer Test

In order to get further speed tests and become familiar with the primitive program API's that are available, a socket based client-server pair of programs were written to test program to program interconnectivity and transfer rates. Code details are contained under the BENCH_MARK/COMM_TEST directory, which stores the programs written for testing communication rates between machines.

Two programs are of special interest since they serve as the prototype for both TCP/IP STREAM socket programming and the communication speed tests conducted. These two programs are called **client_variable.c** and **server_variable.c** and are listed in Appendix B.

Figure 13 shows a block diagram of the standard UNIX call sequence on both the client and server side required to implement this program to program interface. The program opens a socket between the two machines. A header is sent by the **client_variable()** to specify the size of the data blocks about to be sent. The client then sends a block. The server reads it and echoes it back. The client reads the echo and in turn echoes the received data back to the server. Both programs Ping-Pong the data back and forth a *number_of_blocks* times. After the loops are completed the client checks the data content and prints out the time elapsed and number of errors occurring in the overall transmission. The programs must be run in pairs with the server started first on one machine and the client started second. The server waits at the "listen" until the client tries to connect.

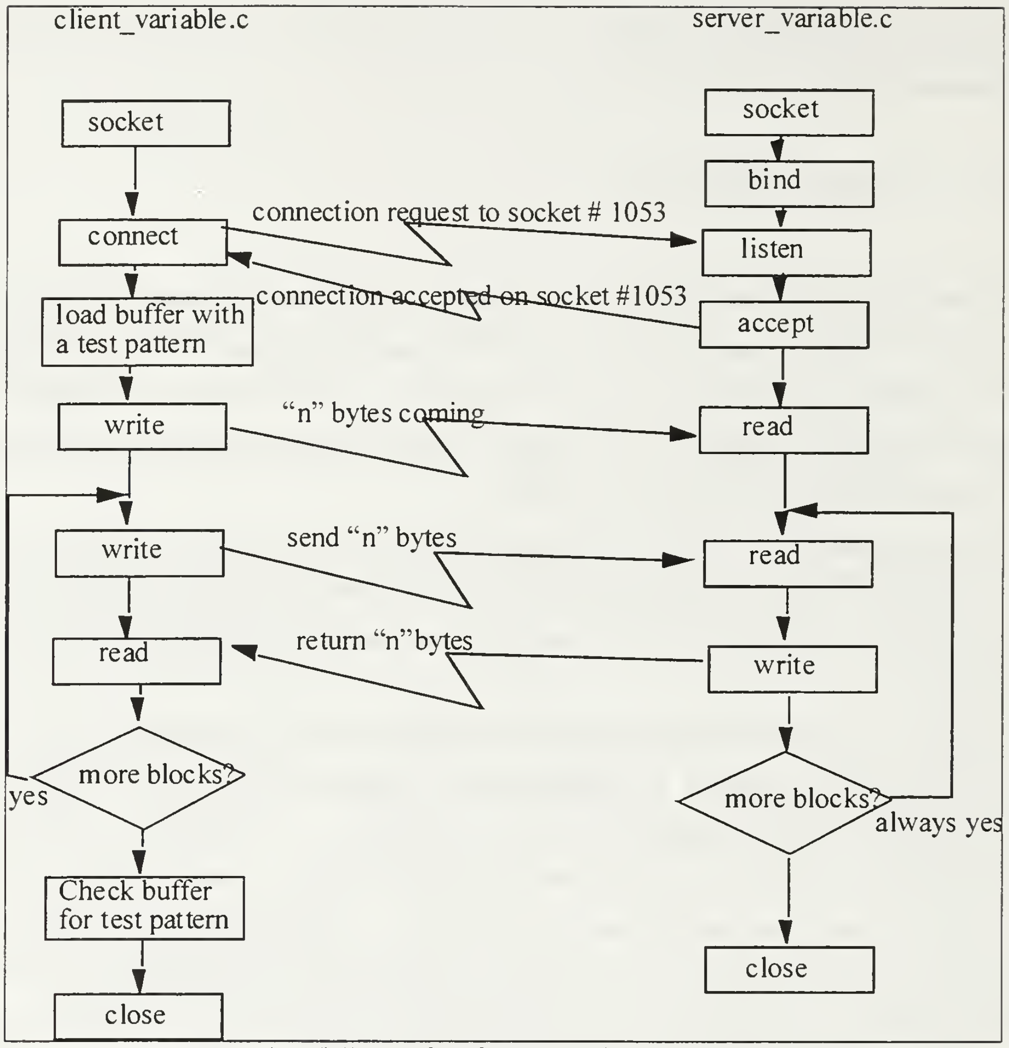


Figure 13. UNIX TCP Calls for Socket Connection

Communication speed tests involve both raw hardware speeds as well as the setup and error checking conducted by the communications (in this case TCP/IP) software layers. The overall transfer speeds therefore depend upon the block size being sent since communication software overhead is largely independent of the message size. Transfer rates were calculated by dividing the total number of bytes transferred back and forth by the total amount of real time to do so.

The client server program pair was run as both single and simultaneous parallel processes. Configuration of these processes and the network hardware accessed are shown in Figure 14. Each server is started listening to a unique port number. The clients then attempt to establish a connection to these port numbers. In addition to port numbers, the clients specify the network name of the server machine. Hence, as shown in figure

14, the clients connect through **wolfpc**, **wolfpc2**, or **wolfpc3** to distinguish between the connection hardware desired.

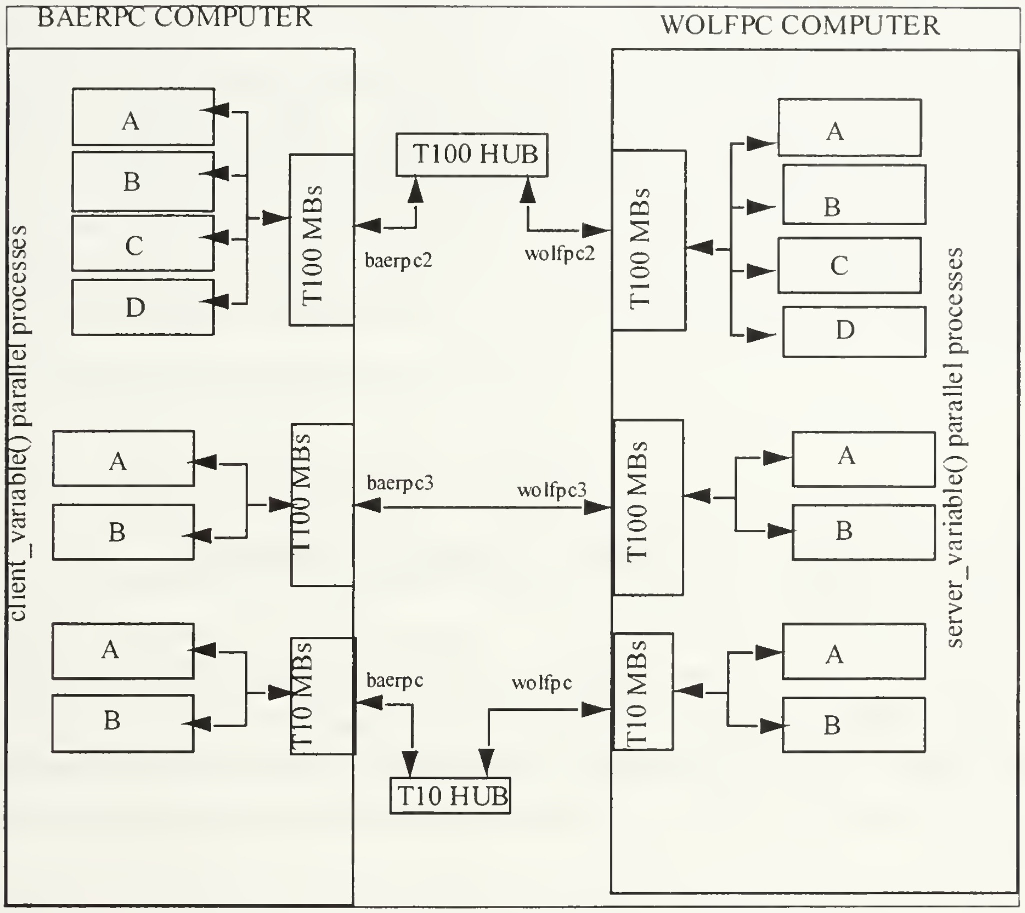


Figure 14. Process and Network Configuration

Figure 15 shows the cumulative transfer rates and the percent of idle CPU time as a function of the process and connectivity configuration exercised. The processes running are specified by the **xs** in the columns on the left. For example, the first three lines indicate only one process was active on the T10, T100 hub, and T100 direct connection lines at a time. The fourth line indicates two processes were active on the T100 Hub connection.

processes combinations running								BLOCK SIZE IN BYTES				%CPU
T100 Hub				T100 direct		T10		200	600	1000	1400	IDLE
A	B	C	D	A	B	A	B					
						x		0.31	0.45	0.49	0.51	80%
x								0.52	1.08	1.34	1.52	77%
				x				0.49	1.08	1.36	1.55	77%
x	x									1.75	2.05	69%
x	x	x								2.00	2.31	66%
x	x	x	x							2.13	2.47	62%
x				x				0.55	1.24	1.80	2.03	54%
x	x			x	x	x	x	0.70	1.53	2.10	2.61	40%

Figure 15. TCP/IP Communication Speed Results

The results show typical efficiency increases as the block size increases. The CPU idle percentage when nothing operated was approximately 95%. During test execution this number was approximately the same for all block sizes so only one number is shown. The numbers clearly show that software efficiency is the biggest resource consumer.

The throughput did not increase when using two hardware devices compared with one (line 4 vs. line 7). The two-process single hardware link provided the most efficient data transfer rates. Though peak performance of 2.5 megabytes per second were achieved using 6 processes on 3 hardware links, the extra half megabyte per second cost approximately 30% of the CPU time while cost associated with the first 2 Megabytes was only 25%.

These results represent an application's program-to-program transfer rate of approximately 15% of the maximum hardware efficiency of approximately 8 Megabytes per second throughput advertised on the 100BT lines. This rate is par for typical Ethernet connections.

Our conclusion from these tests is that substantial performance increases could be achieved on this low cost hardware if special driver code were written. However, if we stick to generally available TCP/IP Ethernet implementations, throughput in the megabyte-per-second range can be expected between point-to-point links. This performance compares favorably with the 0.7 Megabytes achieved on the T800 transputer links used in the parallel processing high resolution battlefield perspective view generator.

An additional lesson can be derived from the test results. The efficiency of multiple hardware connections is not substantial enough to warrant the use of the scarce PC slots or available IRQs. A grid configuration such as supported by the 4-link per node transputer systems are not recommended here. Instead, a single or double string configuration seems advisable. This result would favor the use of multiple CPU nodes such as those available in the Dual and Quad Pentium cards currently on the market to handle extremely compute intensive applications.

5. UDP Communication Test Results

We switched to a connectionless socket UDP protocol and used the **sendto()** and **recvfrom()** functions for communicating. The UDP protocol reportedly increases throughput by a factor of two at the expense of reliability.

Figure 16 shows the measurements made. Throughput increased only by a modest 20%. We did find the transfer was unreliable but errors only showed up after minutes rather than seconds of continuous operation.

processes combinations running								BLOCK SIZE IN BYTES				%CPU
T100 Hub				T100 direct		T10		200	600	1000	1400	IDLE
A	B	C	D	A	B	A	B					
						x		0.35	0.48	0.51	0.53	83%
x								0.68	1.25	1.50	1.68	79%
				x				0.58	1.08	1.57	1.78	79%

Figure 16. UDP Communications Speed Results

Considering the error checking that would be required, there seems to be no major advantage to the UDP protocol on these machines.

6. Additional Communication Test Results

This section documents additional communication tests conducted to explore alternative programming approaches. Most of the results are negative in the sense that little or no performance improvement over the previous results was achieved. Hence their value lies in the lessons learned.

a. Compiler Optimization Tests:

In order to check the effect of compiler optimization we compiled our test program with the maximum **O2** compiler optimization switch as follows:

```
gcc -O2 -o server_variable server_variable.c
gcc -O2 -o client_variable client_variable.c
```

We then sent and received 1000 blocks of 1000 bytes for a total transfer of 2,000,000 bytes with 0 errors in 1.470000 seconds at 1.360544 megabytes per second. This compared with similar transfer rates shown in figure 15 when using 1000 byte size blocks.

We conclude that there does not seem to be any major optimization effect through compiler optimization.

b. Maximum Block Size Test:

Since cumulative performance increases with block size, the question of what is the biggest block size is of interest. Test result printouts show:

```
Sent and received 100 blocks of 1440 bytes with 0 errors.
Total transfer of 288000 bytes with 0 errors in 0.190000 sec at
1.515789 mbs.
Sent and received 100 blocks of 1460 bytes with 0 errors.
Total transfer of 292000 bytes with 0 errors in 0.200000 sec at
1.460000 mbs.
Sent and received 100 blocks of 1520 bytes with 380 errors.
Total transfer of 304000 bytes with 380 errors in 0.510000 sec at
0.596078 mbs.
```

Note that communication is unreliable or fails when the packet size exceeds 1440 bytes. This result corresponds to the limit imposed by the Ethernet packet size. We were not able to eliminate the effect by increasing the TCP buffer size through software. Our conclusion is that STREAM sockets do not in general insulate the user from low-level packet size restrictions and that user level blocking at approximately 1400 bytes or less is required.

7. FDDI Comparison on Other Machines

To get a comparative performance of the STREAM socket interface on other machines we tested the client server pair on the phoenix and ntciris machines at TRAC Monterey. The operating system characteristics of these machines are:

The **uname -a** command returns `SUNOS phoenix 4.1.3_U1 8 Sun4m.`
The **uname -a** command returns `IRIX ntciris s.3 11091810 IP7 mips.`

These two machines are connected with 100 megabit per second FDDI links through a Cabletron Hub. On **ntciris** the FDDI interface card was purchased from Silicon Graphics Incorporated. On **phoenix** a s-bus FDDI 100mbs card from Network Peripherals was used. For 1000 back-and-forth cycles the following test results were recorded:

Block size in bytes -	**200**	**600**	**1000**	**1400**	**2000**	**3000**	**4000**	**5000**
Transfer rate in Mbytes/sec -	.15	.44	.68	.89	1.18	1.41	1.68	error

These rates were 40% slower than the Pentium T100 card systems by when comparable block sizes are used. However, in the FDDI protocol the basic packet size is 4500 rather than the 1500 bytes in a conventional Ethernet. Consequently transmission errors did not occur until block sizes in the neighborhood of 4000 bytes were sent and the data rates were higher at the large block size limit of the FDDI system.

This experiment demonstrates that large size packets on STREAM sockets do not transmit reliably unless the application performs handshake and blocking. It also showed that the performance of the Pentium systems was better than the Sun and SGI workstations they are expected to replace. To check the effect of the software selectable buffer size we executed the following code:

```
sendbuff = 65536;
optlen = sizeof(sendbuff);
if(getsockopt(sock, SOL_SOCKET, SO_SNDBUF, (char *)
&sendbuff,
   &optlen) < 0) perror("getsockopt error");
 printf("default send buffer size = %d\n",sendbuff);
```

The result was default send buffer size = 61440

This limit had no effect on the errors. We note the Ftp transfer of a 296960-byte file across the campus 10mbs link is 233kBytes/s. while the same file transferred between the two machines at 160kBytes/s. These numbers are very usage dependent and only provide a feeling for typical rates expected from reliable large data transmission codes. Clearly the program to program reliable transfer rates appear to be between 10% and 20% of the underlying hardware capability.

8. Reliability Test

To check reliability we ran several of the tests shown in figure 15 for long periods of time. Specifically, the 3 card, 6-process test was run for 16 hours. This represents approximately 1.5 terra-bytes of data with no detected error. Our conclusion is that both the hardware and software is stable as long as the block size is below 1400 bytes and sufficient handshake is implemented by the socket user to eliminate Ethernet buffer overrun.

9. One Way Test

Rather than using an echo scheme, we programmed the client server pair so that one would only write while the other only read. We originally thought was that this would always keep the input buffers full, and we hoped the "advertised" flow control for STREAM sockets would allow the internal software to optimize throughput. To our surprise this resulted in the same type of errors that occurred with large block sizes. Further investigation showed that the client or "write" function calls did not block as specified in the literature. Instead, all writes would execute even if the server corresponding read had been suspended.

At this point the only way we could get reliable one way transmission was to suspend the **write** side for long enough to allow the link to recover. This means the one way transmission rates are approximately half of the numbers stated in Figure 15. It was apparent that the concept of "keeping the hardware pipe full" has not been effective. A communication handshake seems necessary.

10. Message Passing Interface (MPI) and LAM

The Message Passing Interface (MPI) is a networked parallel-processing standard based upon the message passing program paradigm. Several manuals listed in the documentation section describe the C code bindings required to implement this standard. MPI implements a message scheme for multi-process functions to communicate across a network while LAM adds remote host control functions through the use of a supervisory daemon.

The following test results exercised an echo routine similar to the socket algorithm described above only using the **MPI_Send()** and **MPI_Recv()** functions. The time here is the total time for all the back and forth messages to take place in megabytes per second. The total bytes transferred is twice the value shown in the 1way_#ofbytes column and only one half the duplex transmission connection is exercised.

Error	Bytes Sent	Block Size	Messages	Transfer Time	Transfer Rate	CPU Idle
0	640000	200	3200	4.129243	0.309984	57%
0	639600	600	1066	1.759280	0.727116	61%
0	640000	1000	640	1.255434	1.019568	58%
0	639800	1400	457	1.066234	1.200112	58%
0	640000	2000	320	0.972685	1.315945	64%
0	636000	6000	106	0.681736	1.865825	64%
0	640000	10000	64	0.704865	1.815951	64%
0	640000	20000	32	0.608154	2.104730	61%
0	640000	80000	8	0.721946	1.772986	73%
0	640000	640000	1	0.645494	1.982977	34%

Table 1. MPI Round Trip Transfer Performance

The tests conducted indicate transfer rates using LAM are similar to the rates achieved by the direct socket test programs discussed above. For small block sizes the socket programs were faster. However, when block sized above 1400 bytes were given to the MPI functions the resulting transmitting rates were about 25% more efficient. LAM/MPI uses more CPU resources than the socket code. For the 1400 byte block size, the 1.2 megabyte per second rate left only 58% of the CPU idle. The corresponding socket code left 77% of the CPU available for other applications while achieving 1.5mbps-transfer rates.

We also conducted one way transfer tests. In these test the message blocks were sent in one direction and without an echo. The last block was echoed back to allow some error checking. The result were as follows:

Error	Bytes Sent	Block Size	Messages	Transfer Time	Transfer Rate	Send	Recv
0	640200	200	3201	2.599309	0.246296	10	1.8
0	640200	600	1067	0.417554	1.533215	16	2.1
0	641000	1000	641	0.342754	1.870146	23	1.8
0	641200	1400	458	0.309692	2.070444	32	1.9
0	642000	2000	321	0.289548	2.217249	38	2.1
0	642000	6000	107	0.248654	2.581901	48	1.8
0	650000	10000	65	0.340400	1.909518	44	75
0	660000	20000	33	0.301672	2.187807	42	74
0	720000	80000	9	0.337875	2.130966	43	41
0	1280000	640000	2	0.644103	1.987260	36	33 *

Table 2. MPI One Way Transfer Performance

* NOTE: the wall clock time for the 640000 byte transmission was 2 second indicating a substantial amount of time in setup.

The MPI implementation provides reliable one way transmission at essentially the full line rates achieved for the socket implementation. For large blocks the MPI provides a better communication service than direct socket programming and appears to have solved Ethernet packet size limits and one way reliability problems which surfaced in the socket implementation.

Here we also show the %CPU idle for selected test. These were measured by executing the UNIX TOP command repeatedly while sending a continuous stream of messages of the indicated block size. The TOP command shows the percentage of CPU idle as a snapshot of current activity. This figure typically varies from snapshot to snapshot. The numbers shown are averages over several TOP snapshots. The only other process consuming CPU resources at the time was TOP itself which took 2.7% of the processor resources. The results clearly indicate that the LAM/MPI code achieves its performance at the cost of heavy CPU cycle utilization.

Results we have obtained for blocks less than 1400 bytes are consistent with the expectation that small performance penalties are inherent with the extra service provided. Though important for the overall TELLUS development these results primarily verify the validity of the socket implementation discussed in the last chapter. There appear to be reasonably significant faster transfer mechanism implemented inside of these more sophisticated MPI packages which would speed up the application software level communication rates.

The emergent MPI as a platform independent standard together with these performance results indicate that MPI is a preferred implementation API for distributed processing applications at this time.

B. IMAGE DISPLAY TEST

A simple **X Window** based routine was written which generates two 256x256 image buffers in main memory. The program then calls the **XputImage()** routine to alternately write the images to the window. This tests the time it takes to display a perspective view image once the content is calculated.

Figure 7 shows the basic timing loop at the top. This calls **UpdateDisplay()** which displays the content of the buffer passed to it.

```
                              *
                              *
                              *
     puts ("PRESS RETURN TO START BENCH MARK...");
    getchar ();

    window = (WINDOW *)
          DisplayPixels (array[0], 256, 256, colortab, 64, "",
          argc, argv, bboard);

    XtAddCallback (window->pd, XtNdestroyCallback, (XtCallbackProc) exit, 0);

    a = time(0);
    for (i = n = 0; i < 100; i++, n = !n)
      UpdateDisplay (array[n], window);

    b = time(0) - a;

    printf ("Total time took %d seconds to display 100 frames\n"
          "Average frame rate was %d per second\n", b, (int) (100 / b));

    exit (0);

    XtRealizeWidget (toplevel);
    XtAppMainLoop (app);
}
```

call

```
void UpdateDisplay (unsigned char *pixels, WINDOW *window)
{
  register int x, y, w, h;

 w = window->image->width;
 h = window->image->height;
 window->image->data = pixels;

 XPutImage(XtDisplay(window->w), XtWindow(window->w), window->gc,
        window->image, 0, 0, 0, 0, w, h);
}
```

Figure 17. Image Display Benchmark

The following results were noted during execution of the **XputImage()** routine. As you can see, Pentium based PC has the potential to provide smooth video at a substantial frame rate. Recently released inexpensive video cards promise even faster frame rates.

Platform	Video Performance
TRAC SUN (phoenix) dedicated running under Openwin	30 fps
Sparc station IPX networked	15-20 fps (load dependent)
Pentium 100 with an S3-968 based PCI video card	50 fps

Table 3. Frame rate comparisons

C. CPU AND MEMORY TEST

A standard ray trace benchmark (Appendix C) was tested on a series of machines. The program sets up a 1000x1000-element terrain array with elevations set at 500 meters and performs 50,000 ray trace calculations from 1000-meter altitude to 500 in 5 meter steps. The total number of inner loop steps is 50 million. The following table was published (Baer, 1991) and is duplicated here along with additional tests performed on TRAC and NPS machines.

Processor	Language	Witness	Rate
Sun 4/260	C	Nascent	311
Sun 4/260 16Mhz	C -O3	Nascent	56
IRIS 4D/70GT R2000 12.5Mhz	C	Nascent	243
IRIS 4D/70GT R2000 12.5Mhz	C -O3	Nascent	119
SGI R3000 25 MHz	C	IDC	49
T800 25Mhz	OCCAM	Parsytech	164
Alliant FX/80	C	IDC	197
i860 33 MHz	C	Intel	27
SGI PowerSeries(ntciris)	C	Baer	128
SGI PowerSeries	C –O	Baer	93
SUN (phoenix)	C	Baer	85
SUN (phoenix)	C -O3	Baer	41
SGI 4 processor Onyx(ntciris)	C	Baer	56
SGI 4 processor Onyx(ntciris)	C –O	Baer	48
Pentium 90 MHz under Linux	G++	Baer	40

Dual Pentium 133mhz SMPlinux	G++	Decato	10
Dual Pent-PRO 200mhz Win-NT	c++	Baer	1.25

Table 4. CPU and Memory Benchmark Comparison

Based on these benchmarks, the only processor faster than the Pentium-90 was the i860. These tests on the i860 were conducted at Intel after a similar test conducted by IDC showed performance numbers of more than 80 seconds. We know Intel engineers ran the code exactly as defined but were not able to get other details (i.e. memory speed, etc.) to fully understand the very high speed this test produced.

The Pentium PCs under Linux using the free GNU g++ compiler are as fast or faster than any of the workstations on which the PEGASUS battlefield simulation has been run. We should note that although the SGI (ntciris) had four processors, the program only executed on one of them. The dual-processor Pentium-pro 200-megahertz machine is extremely fast for our application.

1. Effect of Memory Speed and Cache

The effect of CPU speed does not seem to be the determining factor in the performance result of the ray trace benchmark.

The statement

```
e += de;
```

increments the ray tip position across columns. When run on the Pentium 90, execution time was reduced to an amazing 4 seconds. This is identical to the time the program runs if all main memory access is eliminated by taking out the array reference statement:

```
zt = terrain[uwe][uwn];
```

Hence we conclude that 90% of the execution time is spent on main memory access. Increasing the size of the cache or purchasing faster memories would be a more effective speed up than simply getting more CPU cycles by buying increased clock speed machines.

The effect of additional local DRAM memory hits was explored by adjusting the benchmark routine so the terrain array was organized in blocks instead of rasters. This was accomplished by replacing the bit shift statements above the terrain reference by the following routine:

```
e1 = (e>>16) & oxf;
n1 = (n>>12) & 0xf0;
uwl= n1 | e1;

e2 =(e>>20) &0xf;
n2 =(n>>16) &0xf0;
e3 =(e>>16) &0x300;
n3 =(n>>14) &0xc00;
uwh = n3 | e3 | n2 | e2;
zt =terrain[uwh][uwl];
}while(zr > zt);
```

Figure 18. Extract Upper 16 Bit Coordinates

This code runs in 20 seconds with the modification--twice as fast as the raster access code in the standard benchmark. This example demonstrates how performance can be dramatically improved by simply replacing the two lines of code executing two shift operations with eight lines of code executing 17 bit operations. This information is of great importance both in guiding the writing of fast code as well as making purchasing decisions. We speculate that the Pentium Pro design, which includes a 256kb cache on each processor in the dual configuration, is fast because of this cache effect, not the clock speed.

2.- Frame Rate Performance Predictions

The CPU performance benchmark is indicative of the frame rate achievable from a multi-processing system executing the inverse ray trace PVG algorithm of the PEGASUS/TELLUS projects.

Table 5 compares the benchmark results on various machines in which the simulation has been implemented and actual frame rates measured.

Machine	**Speed In seconds**	**# Processors**		**Frame Rate**
T800 25mhz	164	16	10	4
T800 and Power PC 601	40 (est)	21	2	16
SGI PowerSeries	93	1	93	.5
SUN (phoenix)	41	2	20	1
Pentium 100 under Linux	40	1	40	1.4
Dual Pentium 133 SMP Linux	20	2	10	6
Dual Pentium Pro 200 under Windows NT	2.5	2	1.25	30

Table 5. Benchmark and PVG Frame Rate Comparison

The relationship between the system speed for the benchmark and the frame rate achieved in an implementation is fairly predictable at one frame-per-second for 20 seconds of system benchmark performance. This would indicate a dual Pentium Pro could theoretically be able to generate at least thirty frames per second. However, video display and disk access rates will limit the actual frame rate to approximately 15 frames per second or less.

Our findings indicate that the TELLUS PVG will be able to generate about 10 frames per second for a 256x256 pixel-sized frame. At this rate, sufficient CPU capacity should be available to include several high-resolution targets. The bottleneck will be disk access and video display speed.

IV. SUMMARY, RECOMMENDATIONS, AND CONCLUSIONS

A. SUMMARY

The personal computer is now powerful enough to handle computationally challenging tasks associated with perspective view generation. When properly configured with the high-speed input/output devices, performance rivals that of the UNIX workstations considered "state-of-the-art" just two years ago. Additional performance can be leveraged from systems using a parallel approach that includes multiple processors on a single machine. However, Fast Ethernet performance still does not provide enough bandwidth to transfer uncompressed video over the network at a usable frame rate.

When cost is a factor, system architecture built around Intel's Pentium processor provides the best value. The Pentium's lack-luster floating-point performance may hinder computationally intensive applications and justify a move to a more expensive Alpha based platform.

Many communications libraries exist to ease development of message passing strategies. TCP/IP sockets and streams remain the most portable and well documented of the libraries we examined. The Message Passing Interface, however, is an emerging standard that provides robust communications interfaces and reliable cross-platform communications. MPI is simple to use and eases communications intensive software development.

B. CONCLUSIONS

Our research concludes that an Intel Pentium equipped Personal Computer is a worthy platform suitable for applications such as the PEGASUS Perspective View Generator. The current Pentium Pro provides ample floating-point and integer performance to deliver POV ray trace calculations at reasonable speeds. The Linux operating system, coupled with an X-Windows Motif port, offers an affordable alternative to proprietary and often expensive commercial versions of UNIX. Because Random Access Memory (RAM) performance continues to lag behind CPU performance, bottlenecks exist between the CPU, the memory bus, virtual memory and the hardware bus. Even with the advent of 100BT Ethernet cards, any architecture that attempts to share memory over the network using message-passing protocols adds another choke point to the equation.

C. RECOMMENDATIONS

A distributing approach for further development effort on the PEGASUS Perspective View Generator is not recommended due to bandwidth constraints of Fast Ethernet. A parallel approach using local memory on a multi-processor platform offers greater promise of performance improvements related to real-time terrain modeling simulations. Further exploration of the availability and accessibility of hardware solutions designed to improve upon the limitations of the 33-megahertz PCI Bus speed is warranted.

D. ANSWERS TO RESEARCH QUESTIONS

To what extent can the PCI bus and new high performance peripherals deliver the performance necessary for battlefield simulation application? Throughput on 100BT Fast Ethernet peaks at 3mb per second with average throughput holding at just under two megabytes per second depending on the packet size. Theoretical advertised transmission rates were not achievable. Bus latency appears to limit performance, but more research must be conducted to identify the limiting factor.

What other performance considerations affect the suitability of the Personal Computer as a prospective platform for high- speed terrain modeling? Ray trace calculations continue to be computationally intensive and a major limiting factor overall frame rate performance. Processors are getting faster, and the Intel's Pentium Pro processors is already capable of exceeding the number of calculations per second achieved on the UNIX workstation considered state-of-the-art just two years ago. Recent advances in personal computer video card technology offer promise of even faster frame rate to enhance Perspective View Generation realism.

Can a distributed approach enhance performance of the Perspective View Generator? A distributed approach does not appear to be worth pursuing at this time. Parallel workload distribution is suitable when the problem can be divided into pieces that require lots of computational power but consume just a small amount of the network bandwidth. Although ray-tracing calculations are easily divisible by terrain or display regions, the volume of resulting data that must be transmitted back to the machine displaying the view will not support adequate frame rates.

E. RECOMMENDATIONS FOR FURTHER STUDY

Linux served as attractive operating system for the TELLUS project in that porting problems from UNIX were minor since Motif served as the common interface. However, most commercial hardware vendors do not provide drivers for their equipment, leaving researchers to either write drivers on their own or wait on the Linux programming community to support the equipment. Windows NT 4.0 appears to be well supported since new hardware is seldom released without drivers for both Windows NT and Windows 95. Both Windows NT and Windows 95 support OpenGL version 1.1.

Since OpenGL has now been ported to the PC, the Perspective View Generator may benefit from its now portable API. Initial tests of lost cost Glint based video cards show as much as a 10 times performance gain over previous 2D cards. Exploring both OpenGL and Microsoft's DirectX API may be valuable since much of the computationally intensive work of generation of the view can be moved from the host CPU to the graphics hardware. Under OpenGL, ray-tracing algorithms may not be necessary since the OpenGL engine can handle perspective view generation. A critical question is whether the new PC based 3D cards can handle the demands of real time terrain modeling.

APPENDIX A. LINUX LAM/MPI INSTALLATION GUIDE

A. WHAT IS LAM?

LAM (Local Area Multicomputer) is an MPI (Message Passing Interface) programming environment and development system for heterogeneous computers on a network. With LAM, a dedicated cluster or an existing network computing infrastructure can act as one parallel computer solving one problem. This paper will discuss some of the current issues affecting parallel computing in general. More specifically, it will present LAM/MPI implantation as a cost effective way to increase computing performance simply by adding low cost machine to Local Area Network.

LAM is a product of the Ohio Supercomputer Center in Columbus Ohio and is implemented on the following platforms:

- Sun (SunOS 4.1.3, Solaris 5.4)
- SGI (IRIX 5.3, 6.1)
- IBM RS/6000 (AIX V3R2)
- DEC Alpha (OSF/1 V3.2)
- HP PA-RISC (HP-UX 10.01)
- Intel X86 (LINUX v1.3.20)

B. WHY USE LAM/MPI?

Since MPI is nothing more than an API for communication machines, why use LAM instead of TCPIP Sockets? While one could certainly achieve the same goal using Sockets, the amount of programming involved is monumental. MPI at its simplest level involves calls to six basic function including the send call on a broadcasting machine and a receive call on the receiving machine. The length and structure of the data being sent can be formatted in user definable types of any size. Socket programming, on the other hand, requires you to keep track of individual packets by limiting the size of a packet to around 1500 bytes. Transmitting packets larger than 1500 requires that some negotiation occur to ensure error-free transmission of the data. During testing of LAM, we consistently sent 640,000 bytes in a single send/receive pair with no errors over a 100-megabit per second LAN.

C. GETTING STARTED WITH LAM UNDER LINUX

The installation of LAM under Linux can completed by following the instructions outlined below:

- Obtain and install the source code for LAM/MPI
- Modify the HOME environment variable to point to the desired destination of the LAM binaries.
- Establish a symbolic link from the appropriate operating system stub file in the *Config* directory to a file called *config*.
- Establish accounts on each machine in your cluster giving each equivalent *rlogin* privileges.
- Establish a LAM based hosts file that lists the names of all machines that will participate in the group/cluster.
- Test your network by running the LAM recon.
- Start LAM running on each machine with the *lamboot* command.
- Write, compile and execute a simple program under the LAM environment!

D. OBTAINING LAM

LAM is available in source code format via FTP from ftp://ftp.osc.edu/pub/lam. Along with the single file containing the baseline source code called *lam60.tar.gz*, there is an additional file containing 17 patches that have to be applied -- *lam60-patch.tar.gz*

.

E. INSTALLING LAM

The files were archived using the UNIX tar program and compressed using gzip to save space. I chose the */usr/src* directory as a place to extract the source files in the following manner:

```
        $ gunzip lam60.tar.gz
```

The resulting file, lam60.tar, must then be extracted as follows using the tar command:

```
        $tar xvf lam60.tar
```

The files will be extracted into a directory called *lam60.*

Once the files have been extracted into the new directory, copy the file *lam60-patch.tar.gz* into the *lam60* directory.

F. APPLYING THE PATCHES

Likewise, extract the patch file as follows:

```
        $gunzip lam60-patch.tar.gz
```

The resulting file, *lam60-patch.tar*, must then be extracted as follows using the tar command:

```
        $tar xvf lam60.tar
```

When this file is extracted, you should have 17 patch files to apply ranging in names from *lam60-patch01* to *lam60-patch17.* Each patch must be applied separately, and in order. For example, you can't install patch 5 before patch 4. To install the first patch, make sure that you are in the lam60 directory and execute the following command:

```
$patch -p0 < lam60-patch01
```

If all goes well on the first patch, you should see the following message:

```
Patching file share/mpi/MPI.c using Plan A…
Hunk #1 succeeded at 4.
Hunk #2 succeeded at 217.
Done.
```

Apply the other 16 patches in the same manner.

G. CONFIGURING LAM FOR INSTALLATION

The only instruction document for installing LAM (other than the document you are reading now) is located in the */lam60/doc* directory. Unfortunately, the file is in HTML format, so if you attempt to view it, you'll have to deal with the HTML formatting characters. The actual file name for this help file is *lam-install.html.*

The */lam60/Config* directory contains copies of all the stub files necessary to get LAM to compile with the appropriate operating system. Change to the *./lam60/Config* directory. By default, the LAM installation process installs a stub file to the Sun Operating System. Remove the existing *config* file (which is just a symbolic link) by typing:

```
$ rm -rf config
```

Now it is safe to establish a symbolic link the Linux stub file as follows:

```
$ ln -s config.i386_linux config
```

Once the symbolic link file *config* has been created, open it for editing. You'll notice near the top of the file there is an entry referring to the *Environment Variable* HOME.

The line should look something like this:

```
HOME=   /tmp/lam
```

This entry tells the LAM compilation process where to put the executable binaries once the files have compiled. You should edit this directory entry to reflect your preferred directory where you want the files placed.

H. COMPILING LAM

Build LAM by changing to the *lam60* directory and type *make*. To run *make* in the background, execute the following command:

```
$ make >& LOG.TXT &
```

This forces Linux to run make in a background process and capture all output to a file called LOG.TXT. At the end of the compile, you can check the contents of the LOG file to make sure you had a successful compile.

When making the executable in this manner, you will get no visible feedback that the compile is actually working. I suggest you monitor its progress with "top". When the compilation is complete, examine the LOG.TXT file to be sure that no errors have occurred. The actual compile process can be surprisingly slow (10 minutes on my 100 MHz machine and 6 minutes on my 133 MHz machine) as there is lots of code to compile (120+ object files are created). That's almost as much time it takes to compile a kernel!

I. POTENTIAL PROBLEMS WHEN COMILING

We recommend installing LAM with the latest version of the official release of Linux, which currently is version 2.0.20. However, if you happen to have an earlier version of the gnu C compiler and an earlier version of Linux you may encounter the following error during the compilation process:

```
In file included from ../../../share/mpi/rpi.c2c.c:33:
/usr/include/sys/uio.h:  redefinition of 'struct iovec'
make[2]: ** [rpi.c2c.o] Error 1
```

There are two system files that cause this problem. To correct the error you should either obtain the latest version of Linux and the GNU C compiler or modify the affected files.

The two files are: */usr/include/sys/uio.h* and */usr/src/linux/include/linux/uio.h.* If you examine the latter file, you will see a warning hidden away as a comment

/* A word of warning: Our uio structure will clash with the C library one (which is now obsolete). Remove the C library one from sys/uio.h if you have a very old library set */

We can only guess what defines a "very old library set". Suffice it to say that I had to manually modify my files until I installed the newest version of the sources. To correct this problem without going through the trouble of installing a new C library, you can edit the files as follows:

- Remove the definition for struct iovec in /usr/include/sys/uio.h
- Include the file "/usr/src/linux/include/linux" with the includes at the top of the same file.

This replaces the original definition for the structure iovec with the Linux unique version of the structure.

J. RUNNING LAM

LAM requires *rlogin* access between each of the machines it will be running on. The easiest and safest way to do this is to create a *.rhosts* file in the home directory of the account you've create for LAM on each machine. For security reasons, this file must be owned and created by the superuser (System Administrator) but should be readable by all. The format for the *.rhosts* file is:

```
[hostname] <user>
```

Let's examine a sample *.rhosts* file that LAM could use. Assume that we have 5 Linux machines in our network named **wolfpc, baerpc, lionpc, pumapc, and owlpc.** On each machine we've established a LAM account. Each account must have the same login name and password. Each machine must have LAM installed and the accounts must be able to access the LAM binaries (the lam/bin directory must be in the default path).

The sample *.rhosts* file for this scenario looks like this:

```
wolfpc
baerpc
pumapc
tigerpc
owlpc
```

We can check the file permissions for our sample .rhosts file:

```
$ ls -la .rhosts

rwxr..r..r.. .rhosts
```

All users from each of these machines are granted access to their accounts without a password check.

Prior to running LAM you should attempt to rlogin to each of the machines that will be sharing on the LAM network. Log in as the LAM user on one machine and attempt to rlogin to all the other machines. If you are prompted for a password, either the permissions are set wrong on the *.rhosts* file or there is no *.rhosts* entry for your host on the other machine.

Another important element to setting up your LAM system is to be sure that the LAM binaries (typically installed in the /tmp/lam/bin directory) are in the path of the LAM users you have established.

Another easier, but less secure way to establish host equivalency for lam on your network is to add the host names to the /etc/*hosts.equiv* file. For some reason you have to add the fully qualified host.domain-name as well as just the host name to the file. Each entry must also be on separate lines.

In our example, the hosts.equiv file would look like this:

```
# Sample hosts.equiv file

wolfpc.mbaynet.com
wolfpc
baerpc.mbaynet.com
baerpc
pumapc.mbaynet.com
pumapc
tigerpc.mbaynet.com
tigerpc
owlpc.mbaynet.com
owlpc
```

K. IDENTIFYING THE LAM GROUP PARTICIPANTS

Once you have the binaries installed on each machine, and have rlogin equivalency established you are ready to test the network. However, you must create a file that lists the names of the hosts who are allowed to participate in the LAM network. A sample file called *host.lam* was installed in the */tmp/lam/boot* directory. The name of the file is important since it is passed as a command line parameter when you start LAM running in the background. Simply list the host name of each machine that will participate.

Now you are ready to give it a try!

L. THE RECON COMMAND

To test your configuration, type the following command

```
recon -v /tmp/lam/boot/host.lam
```

The *recon* command attempts to log in to each of the hosts listed in the host.lam file and test them to see if in fact the lam daemon can be executed.

```
# Sample LAM host file

wolfpc
baerpc
pumapc
tigerpc
owlpc
```

The most common reasons for an error to occur during *recon* are:

- **No equivalent rlogin privileges for the host specified**. If this occurs, check the hosts.equiv file to be sure you've included an entry for the fully qualified host name as well as the shorted host name only.
- **Incorrect path on the tested host**. If the LAM recon utility can't execute the LAM binaries from the user's home directory it will fail. In this case you must modify the csh.login file to be sure you've added the appropriate

statement to the environment path so LAM can get to the binaries. If all goes well, your output will look something like this on the originating machine:

```
recon: testing n0 (wolf)
recon: testing n1 (baer)
recon: testing n2 (tiger)
recon: testing n3 (puma)
recon: testing n4 (owl)
```

M. BOOTING LAM

Once you've successfully executed recon (with no error messages returned), you are ready to boot LAM. Booting gets the LAM daemon running on each of the machines participating in the group.

The boot command is executed as follows:

```
$ lamboot -v /tmp/lam/boot/host.lam
```

When this command executes, it displays the following message:

```
LAM 6.0 - Ohio Supercomputer Center


hboot n0 (wolfpc)...
hboot n1 (baerpc)...
hboot n2 (tigerpc)...
hboot n3 (pumapc)...
hboot n4 (owlpc)...
topology done.
```

When you see the "topology done" message, you know you are in business!

N. COMPILING PROGRAMS USING LAM

If you've made it this far you are probably want to get to work writing your programs. I recommend you at least compile and run the sample program *ezstart.c* provided with LAM in the *./lam60/examples* directory.

To compile the program, execute the following command:

```
$ hcc -o ezstart ezstart.c -lmpi
```

Notice that the LAM folks, to make compilation simple, have provided the hcc command. It magically compiles and links your code without a lot of fuss.

APPENDIX B. TCP/IP SOCKET BENCHMARK PROGRAM

```
/**********************************************************************
*
 * FILENAME:server_variable.c
 * PURPOSE: Test the socket communications to a remote host by
receiving
 *            any number of  blocks of variable length bytes
 *            the length must be sent in the first transmission
 *            and sending a reply

**********************************************************************
/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define PORT_NUMBER 1053
#define MAX_BLOCK_SIZE 65536
#define MAX_NUMBER_OF_BLOCKS 65536
#define MESSAGE_HEADER_SIZE 8
#define VARIABLE_HEADER_TAG 314159

main(argc, argv)
     int argc;
     char *argv[2];
{
     int message_header_size;
     int header_body[2];
     int message_body_size;
     int message_body[MAX_BLOCK_SIZE];
     int bytes_read;
     int sock;
     int msgsock;
     struct sockaddr_in server;
     int backlog, serverlen;
     int bindstat;
     int j, comm_error, k, i = -1;
     int diagnostic_switch;

     /* check on usage */
     if (argc >= 3)
     {
          fprintf(stdout, Usage > server_variable[diag_switch] \ n ");
          exit(0);
     }
     /* decode diagnostic switch if present */
     if (argc == 2)
     {
          sscanf(argv[1], %d, &diagnostic_switch);
     } else
```

```
    {
         diagnostic_switch = 0;
    }

    /* open server  socket */
    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    if (sock < 0)
    {
         perror("opening stream socket ");
         exit(1);
    }
    /* Set up parameters for socket connection */
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons(PORT_NUMBER);

    /* bind to server socket */
    bindstat = bind(sock, (struct sockaddr *) & server, sizeof server);
    if (bindstat < 0)
    {
         perror(binding stream socket);
         exit(1);
    }
    /* listen to socket */
    backlog = 5;

 /* The backlog parameter defines the maximum
  * length the queue of  pending  connections
  * may  grow  to. */

    listen(sock, backlog);

    /* notify operator of server ready to accept socket */
    printf("Server ready to accept socket.^C to exit.\n");

    /* accept socket connection and get receive socket number */
    serverlen = sizeof(server);
    msgsock = accept(sock,
                              (struct sockaddr *) & server, (int *)
&serverlen);
    if (msgsock < 0)
    {
         perror("accept ");
         exit(2);
    }
    /* receive the header message */
    message_header_size = MESSAGE_HEADER_SIZE;  /* bytes in header */
    if (read(msgsock, header_body, message_header_size) < 0)
    {
         perror("receive message on stream socket ");
         exit(1);
    }
    if (header_body[0] != VARIABLE_HEADER_TAG)
    {
         printf("Variable header tag missmatch.Wrong program pair.\n");
```

```
        exit(1);
    } else
    {
        message_body_size = header_body[1];
    }

    /* notify operator of ready server */
    printf("Server starting read loop of %d byte blocks. ^C to
exit.\n",
                message_body_size);


    while (1)                       /* start endless communication loop */
    {

        /* receive the message body */
        if (read(msgsock, message_body, message_body_size) < 0)
        {
            perror("receive message on stream socket");
            exit(1);
        }
        /* diagnostic of  communication loop ******************** */
        if (diagnostic_switch < 0)
        {
            i = i + 1;
            comm_error = 0;
            for (j = 0; j < (message_body_size / 4); j++)
            {
                if (message_body[j] != j)
                    comm_error = comm_error + 1;
            }
            printf("block %d size %d error %d \n",
                    i, message_body_size, comm_error);
        }
        if (diagnostic_switch > 0)
        {
            for (j = 0; j < diagnostic_switch; j++)
            {
                k = j - 10;
            }
        }

    /****************************************************************/

        if (write(msgsock, message_body, message_body_size) < 0)
        {
            perror("Sending message on stream socket ");
            exit(1);
        }
    }                                           /* end while loop */

    /* close the socket */
    if (close(msgsock) < 0)
    {
        perror("Socket close error ");
```

```
            exit(1);
        } else
        {
            exit(0);
        }
    }
```

```
/**********************************************************************
 * FILENAME: client_variable.c
 * PURPOSE:  Test the socket communications to a remote host by sending
 *            a "n" blocks of "m" bytes each  and receiving a reply
 *            This routine transmitts the number of bytes in the block
 *            and must be run with the "server_variable" program.

**********************************************************************/
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/time.h>

#define PORT_NUMBER 1053
#define MAX_BLOCK_SIZE 65536
#define MAX_NUMBER_OF_BLOCKS 100000000
#define MESSAGE_HEADER_SIZE 8
#define VARIABLE_HEADER_TAG 314159

main(argc, argv)
     int argc;
     char *argv[2];
{
     int size_of_block;
     int number_of_blocks;
     int message_header_size;
     int header_body[2];
     int message_body_size;
     int message_body[MAX_BLOCK_SIZE];
     int sock;
     struct hostent *remote_host, *gethostbyname();
     struct sockaddr_in server;
     int connect_stat;
     int i, j, k;
     int diagnostic_switch;
     int comm_error;
     struct itimerval timevalue, oldtimevalue;
     long timedif_sec, timedif_usec;
     float timedif;
     /* check on usage */
     if ((argc != 4) && (argc != 5))
     {
          fprintf(stdout,
                    "Usage > client_variable remote_host
number_of_blocks \n ");
          fprintf(stdout, "size_in bytes[diag_switch] \n");
          exit(0);
     }

     /* decode diagnostic switch if present */

     if (argc == 5)
     {
          sscanf(argv[4], "%d ", &diagnostic_switch);
     } else
     {
          diagnostic_switch = 0;
     }

     /* get block size and number of blocks from command line */
     sscanf(argv[3], "%d ", &size_of_block);

     /* here is the block size in * bytes */
     sscanf(argv[2], "%d ", &number_of_blocks);
     size_of_block = size_of_block / 4;    /* convert to integers */
     if (size_of_block > MAX_BLOCK_SIZE)
     {
          printf("Block size should be less than %d bytes.\n ",
MAX_BLOCK_SIZE * 4);
          exit(1);
     }
     if (number_of_blocks > MAX_NUMBER_OF_BLOCKS)
     {
          printf("Number of blocks should be less than %d.\n ",
MAX_NUMBER_OF_BLOCKS);
          exit(1);
     }
     /* open client socket */
     sock = socket(AF_INET, SOCK_STREAM, 0);
     if (sock < 0)
     {
          perror("opening stream socket ");
          exit(1);
     }
     /* Set up parameters for socket connection */
     server.sin_family = AF_INET;

     /* check to see if this is a valid host */
     remote_host = gethostbyname(argv[1]);
     if (remote_host == 0)
     {
          fprintf(stderr, "%s:unknown host \n ", argv[1]);
          exit(2);
     }
     bcopy((char *) remote_host->h_addr, (char *) &server.sin_addr,
               remote_host->h_length);

     server.sin_port = htons(PORT_NUMBER);

     /*-connect to server socket */
     connect_stat = connect(sock, (struct sockaddr *) & server, sizeof
server);

     if (connect_stat < 0)
     {
          perror("connecting stream socket ");
          fprintf(stderr, "Possible problems are:\n ");
```

```
        fprintf(stderr, "Did you start the server program on the other
machine ? \n ");
        fprintf(stderr, "Big - little endian may be reversed for
port_number.\n ");
        fprintf(stderr, "Trying using[server.sin_port =
htons(port_number)].\n ");
        exit(1);
    }
    /* build a message content  */
    message_body_size = size_of_block * sizeof(int);
    for (i = 0; i < size_of_block; i++)
    {
        message_body[i] = i;
    }

    /* set up an interval timer to have 100000 seconds */
    timevalue.it_value.tv_sec = 100000;
    timevalue.it_value.tv_usec = 0;
    if (setitimer(ITIMER_REAL,
                  (struct itimerval *) & timevalue,
                  (struct itimerval *) & oldtimevalue) < 0)
    {
        perror("Set time ");
    }
    /* now get the current time */
    if (getitimer(ITIMER_REAL, (struct itimerval *) & oldtimevalue) <
0)
    {
        perror("First getitimer ");
    }
    /* send the message header */
    message_header_size = MESSAGE_HEADER_SIZE;
    header_body[0] = VARIABLE_HEADER_TAG;
    header_body[1] = message_body_size;
    if (write(sock, header_body, message_header_size) < 0)
    {
        perror("Writing header on stream socket ");
        exit(1);
    }
    for (i = 0; i < number_of_blocks; i++)
    {
        /* send the message buffer */
        if (write(sock, message_body, message_body_size) < 0)
        {
            perror("Writing message on stream socket ");
            printf("Tried to send % d bytes last integer = %d \n ",
                      message_body[message_body_size / 4]);
            exit(1);
        }
        /* receive a reply */
        if (read(sock, message_body, message_body_size) < 0)
        {
            perror("Reading message on stream socket ");
            exit(1);
        }
```

```
     /* diagnostic of  the communication loop ********************* */
          if (diagnostic_switch < 0)
          {
               comm_error = 0;
               for (j = 0; j < size_of_block; j++)
               {
                    if (message_body[j] != j)
                         comm_error = comm_error + 1;
               }
               printf("block %d size %d error %d \n",
                         i, message_body_size, comm_error);
          }
          if (diagnostic_switch > 0)
          {
               for (j = 0; j < diagnostic_switch; j++)
               {
                    k = j - 10;
               }
          }

     /**********************************************************/

     }                                   /* end communication loop */

     /* now get the current time */
     if (getitimer(ITIMER_REAL, (struct itimerval *) & timevalue) < 0)
     {
          perror("Second getitimer ");
     }
     timedif_sec = oldtimevalue.it_value.tv_sec -
timevalue.it_value.tv_sec;
     timedif_usec = oldtimevalue.it_value.tv_usec -
timevalue.it_value.tv_usec;
     timedif = (float) timedif_sec + ((float) timedif_usec / 1000000);

     /* evaluate the communication */
     comm_error = 0;
     for (i = 0; i < size_of_block; i++)
     {
          if (message_body[i] != i)
               comm_error = comm_error + 1;
     }


     /* print result to terminal */
     printf("Sent and recieved % d blocks of % d bytes with % d
errors.\n ",
               number_of_blocks, size_of_block * 4, comm_error);
     printf("Total transfer of % d bytes with % d errors in % f seconds
at % f mbs.\n ",
               (number_of_blocks) * 2 * size_of_block * 4, comm_error,
timedif,
               ((float) ((number_of_blocks) * 2 * size_of_block * 4) /
timedif) / 1000000.0);
```

```
    /* close the socket */
    if (close(sock) < 0)
    {
        perror("Socket close error ");
        exit(1);
    } else
    {
        exit(0);
    }
}
```

APPENDIX C. RAY TRACE BENCHMARK

```
/*********************************************************************
FILENAME: rtbc.C
PURPOSE: ray trace benchmark code
Tests the speed of the basic inverse ray trace inner loop
**********************************************************************
/
#define MAX_STEP 1000
#define MAX_RAY 50000

main()
{
    float x;
    int terrain [MAX_STEP][MAX_STEP];
    int de,dn,dz,zr,zt;
    int e,n,uwe,uwn;
    int i,j;
    /* initialize variables */

    for( i=0; i < MAX_STEP; i++)
    for( j=0; j < MAX_STEP; j++)
     terrain[i][j] = 500*65536;

     /* initialize direction cosines */

     x = .866 * 65536.0;
     de = (int)(x * 0.866);
     dn = (int)(x * 0.5);
     dz = (int)(.5*65536);
    /*Trace MAX_RAY rays of MAX_STEP meters each */
    for( i=0; i < MAX_RAY;i++)
        {
        /* Initialize trace start conditions */
         zr = 1000*65536;
         e = 0;
         n = 0;
         do
             {
                  /*increment ray coordinates */
                 zr -= dz;
                 e += de;
                 n += dn;
                 /* extract upper 16 bits of coordinates */
                 uwe = e >> 16;
                 uwn = n >> 16;
                 zt = terrain[uwe][uwn];
             } while( zr > zt );
         }
    }
```

LIST OF REFERENCES

Baer, Wolfgang, *Implementation of a Perspective View Generator*, Transputing '91, P. Welch et al. Vol 2, pp 643-656, ISOPRESS, Amsterdam, 1991

Baer, Wolfgang, *Global Terrain Database Design for Realistic Image Sensor Simulation*, 13th DIS Workshop on Standards for Interoperability of Distributed Simulations, Vol I, Sep 18-22, '1995, Orlando, 1995

Booch, Grady, *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company, Menlo Park, 1987

Flynn, M.J.. *Some Computer Organizations and Their Effectiveness*. IEEE Trans. On Computers, vol. C-21, Sept, 1992

ISO/IEC 8652, *Ada 95: The Language Reference Manual & Standard Libraries*, Intermetrics, Inc., Cambridge, 1995

10/100 Mbps NICs, LAN Times Magazine, March 27, 1995

SPEC, *Better Benchmarks*, Standard Performance Evaluation Corporation, Manassas, 1997

Tannenbaum, Andrew S., *Modern Operating Systems*, Prentice Hall, New Jersey, 1992

BIBLIOGRAPHY

Burns, Alan & Wellings, Andy, *Concurrency in Ada*, Cambridge University Press, Cambridge, 1995

Feldman, Michael B. & Koffman, Elliot B., *Ada 95: Problem Solving and Program Design*, Addison Wesley Publishing Company, Reading, 1996

Gropp, William; Lusk, Ewing; Skjellum, Anthony; *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, The MIT Press, 1996

Hennessy, John L. & Patterson, David A., *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., San Mateo, 1990

Hunt, Craig, *TCP/IP Network Administration*, O'Reilly & Associates, Inc., Sebastopol, 1992

Johnson, Howard W., *Fast Ethernet: Dawn of a New Network*, Prentice Hall, New Jersey, 1996

Lewine, Donald, *POSIX Programmer's Guide: Writing Portable UNIX Programs*, O'Reilly & Associates, Inc., Sebastopol, 1991

Pfister, Gregory F., *In Search of Clusters: The Coming Battle in Lowly Parallel Computing*, Prentice Hall, New Jersey, 1995

Rieken, Bill and Weiman, Lyle, *Adventures in UNIX Network Applications Programming*, John Wiley & Sons, Inc, New York, 1992

Shirley, John, Hu Wei, & Magid, David, *Guide to Writing DCE Applications*, O'Reilly & Associates, Inc., Sebastopol, 1994

Snir, Marc, Otto, Steve W., Huss-Lederman, Steven, Walker, David W., Dongarra, Jack, *MPI: The Complete Reference*, The MIT Press, 1996

Stallings, William, *Data and Computer Communications*, Macmillan Publishing Company, New York, 1994

Ward, Rosenberry and Teague, Jim, *Distributing Applications Across DCE and Windows NT*, O'Reilly & Associates, Inc., Sebastopol, 1993

World Wide Web Home page, *Tom's Hardware Guide*, http://sysdoc.pair.com, January 12, 1997

World Wide Web Home page: *The National Imagery and Mapping Agency*, http://www-nmd.usgs.gov, April 2, 1997

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., STE 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dryer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

4. Bert Lundy 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Wolfgang Baer 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Steven Decato 2
   313 Ardennes Circle
   Seaside, CA 93955